Admission Control, We Have a Problem

Ryan Jarvinen, Red Hat

bit.ly/k20ac
sched.co/ekBb

@RyanJ

Developer Advocate, Red Hat OpenShift

bit.ly/k20ac
sched.co/ekBb

Motive:

Sustainable productivity when developing

Goals

Usability Goal:

- A reliable distributed platform that allows me to ***focus on my day job***

Goals for this talk:

1. Understand the primary role of Admission Controllers
2. Understand typical use cases for Admission Control, and when to avoid this topic

What's the problem?

Agenda:

Part 1: Admission Control Basics

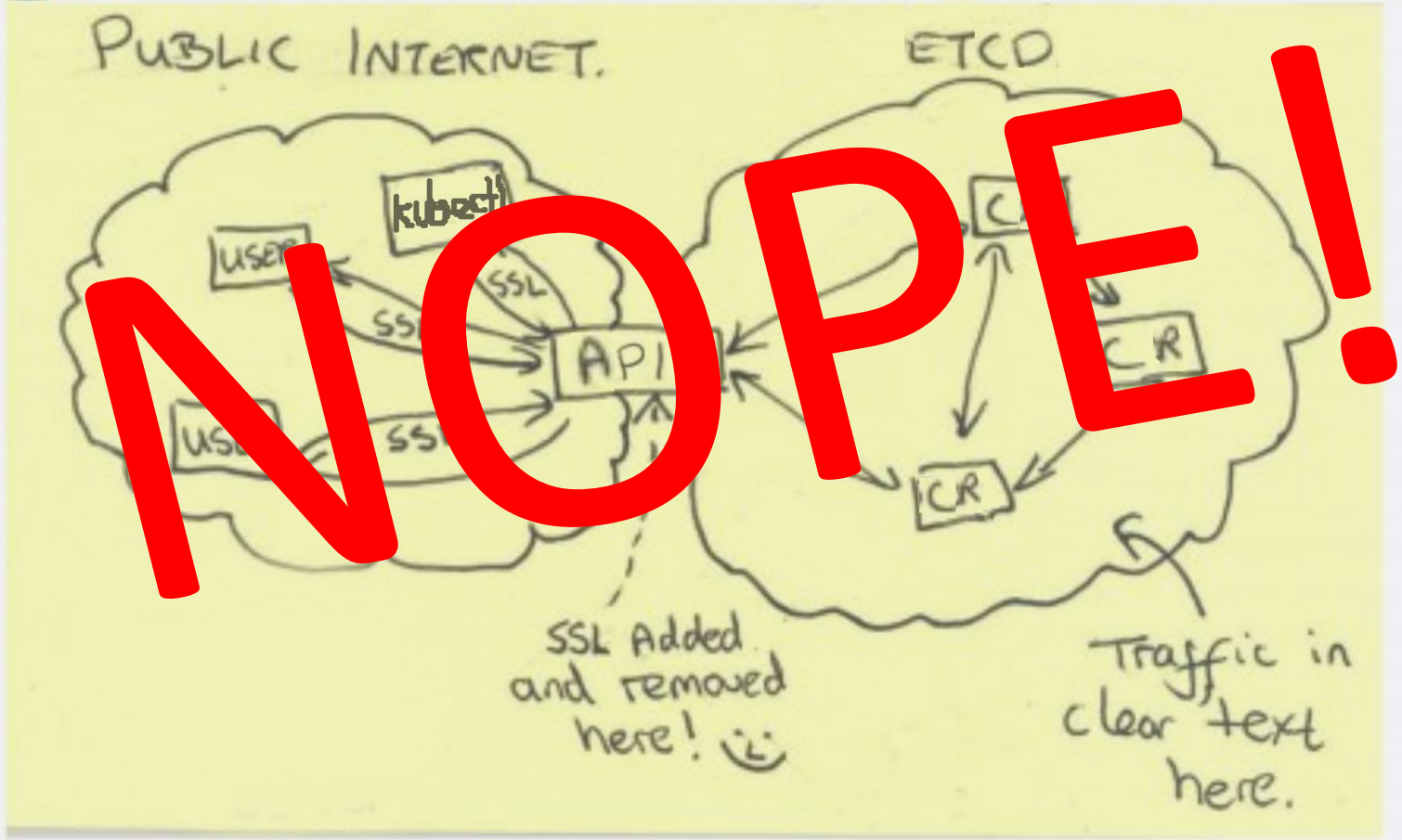
Part 2: Dynamic Admission Control

Part 3: When / How to avoid this topic

Part 1
Admission Control Basics

TOP SECRET//SI//NOFORN

Current Efforts - Google



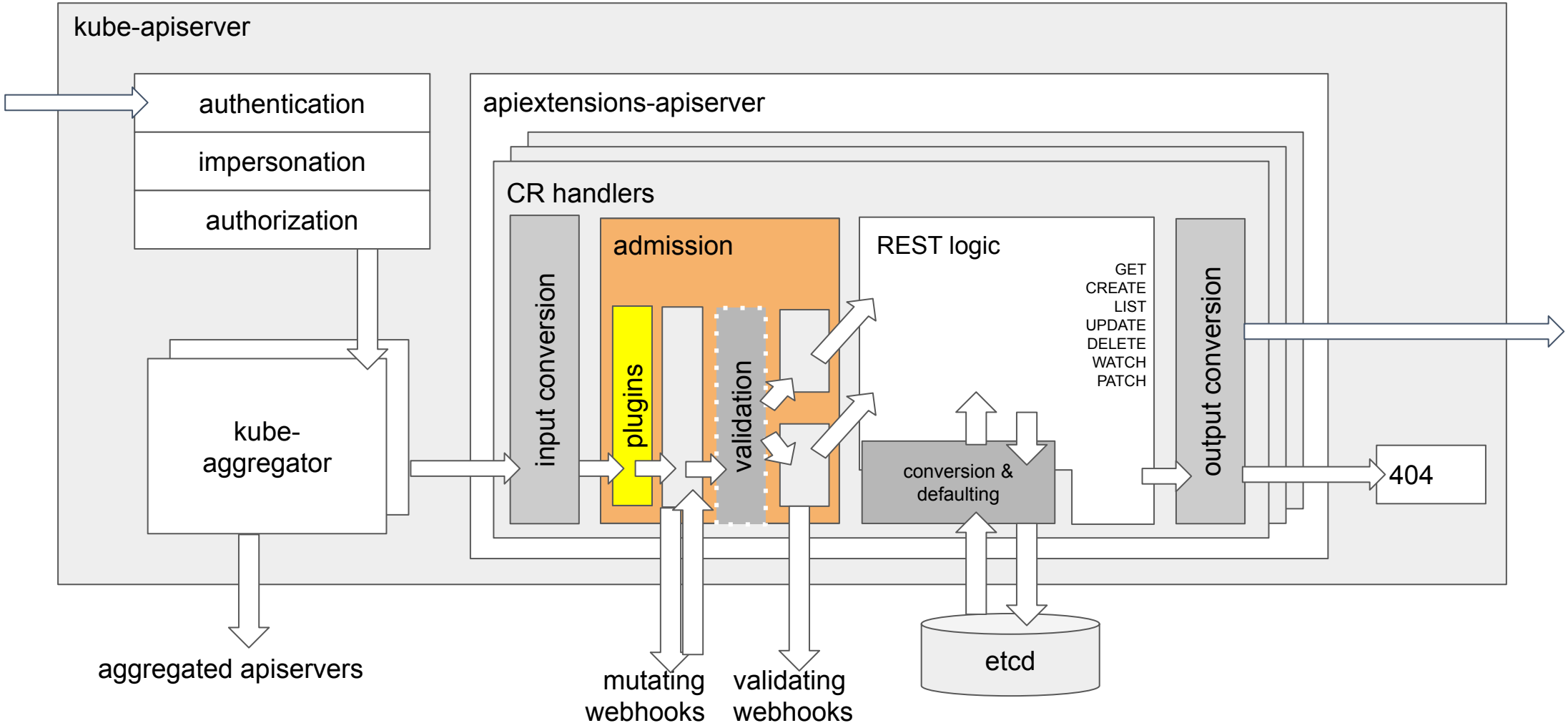
NOPE!

TOP SECRET//SI//NOFORN

ol Plane

Admission Control Basics

Admission Controllers play a critical role in securing the Control Plane:



Admission Control Basics

Docs:
`https://k8s.io/docs/reference/access-authn-authz/admission-controllers`

Q: Wait… Is this why my Operators and CRDs fail to work correctly!?!

A: *Possibly?* Differences in Admission Controller setup are one of the most common reasons why an Operator may fail to work correctly on a cluster

Q: How do I enable / disable Admission Control plugins?

A: Just use the **`--enable-admission-plugins`** and/or **`--disable-admission-plugins`** flags when initializing **`kube-apiserver`**

Q: How do I find out which admission controllers are currently enabled on my cluster?

A: `kube-apiserver -h | grep admission-plugins`

Admission Control Basics

Finding the defaults for your cluster:

minikube w/ API v1.18.2 (kubeadm):

1. `minikube ssh`
2. `ps aux | grep api | head -n 1| sed -e \ 's/.*\(-enable-admission-plugins[^ ]*\) .*$/\1/'`

```
-enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAcc
ount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,M
utatingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQuota
```

OpenShift (v4.5.4 w/ K8s v1.18.3):

1. `kubectl get KubeAPIServers/cluster -o yaml | grep admission -A 10`

```
        admission:
        pluginConfig:
            network.openshift.io/ExternalIPRanger:
            configuration:
                allowIngressIP: false
                apiVersion: network.openshift.io/v1
                kind: ExternalIPRangerAdmissionConfig
            network.openshift.io/RestrictedEndpointsAdmission:
            configuration:
                apiVersion: network.openshift.io/v1
                kind: RestrictedEndpointsAdmissionConfig
```

AlwaysPullImages

This admission controller modifies every new Pod to force the image pull policy to Always. This is useful in a multitenant cluster so that users can be assured that their private images can only be used by those who have the credentials to pull them. Without this admission controller, once an image has been pulled to a node, any pod from any user can use it simply by knowing the image's name (assuming the Pod is scheduled onto the right node), without any authorization check against the image. When this admission controller is enabled, images are always pulled prior to starting containers, which means valid credentials are required.

AlwaysDeny

FEATURE STATE: `Kubernetes v1.13 [deprecated]`

Rejects all requests. AlwaysDeny is DEPRECATED as no real meaning.

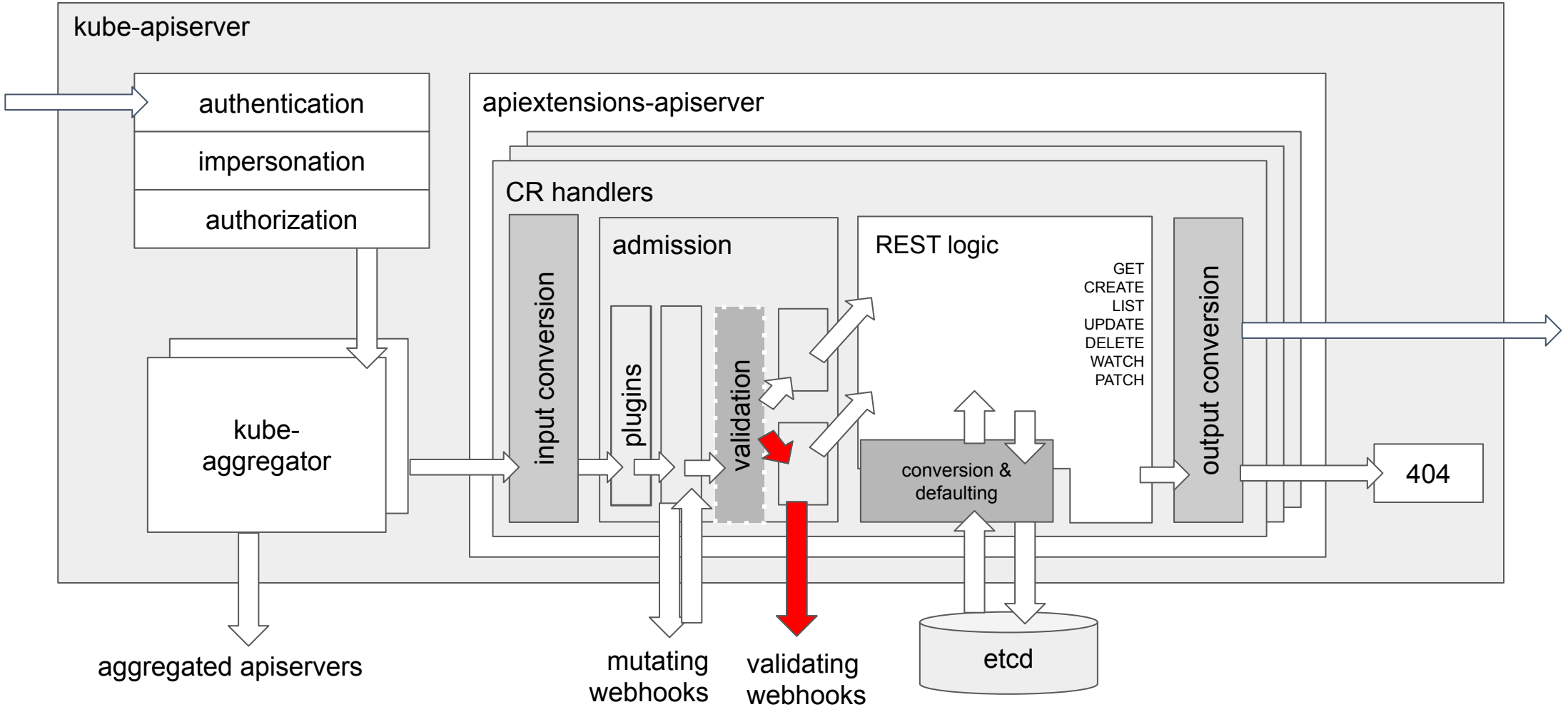
CertificateApproval

This admission controller observes requests to 'approve' CertificateSigningRequest resources and performs additional authorization checks to ensure the approving user has permission to `approve` certificate requests with the `spec.signerName` requested on the CertificateSigningRequest resource.

See Certificate Signing Requests for more information on the permissions required to perform different actions on

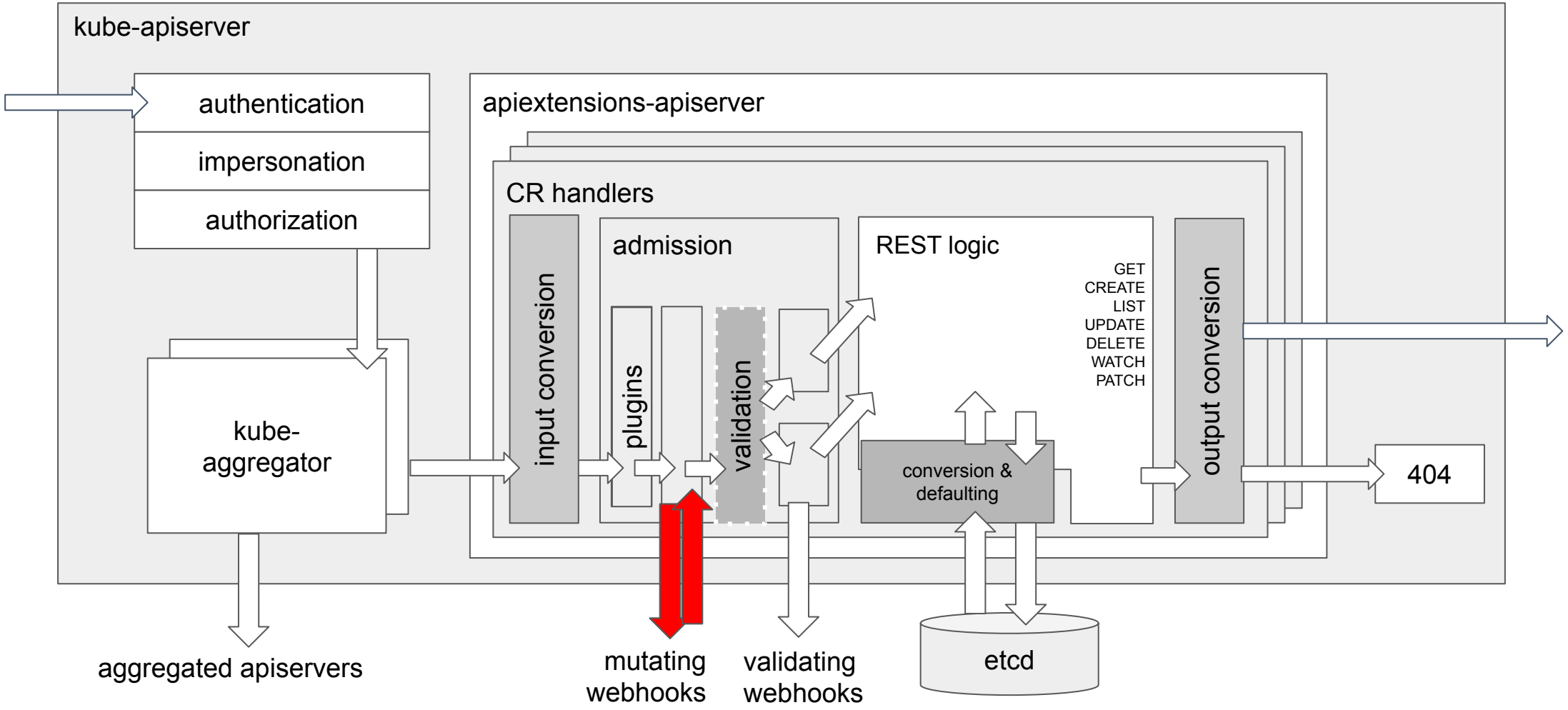
Admission Control Basics

Admission controllers may be "**validating**", "**mutating**", or both!



Admission Control Basics

API writes can be automatically rewritten or coerced:



Admission Control Metaphor

Admission Controllers are similar to Kernel Modules in that they:

- 💥 Operate with elevated privilege scope
- 🔐 Are best configured by a system admin
- ⛔ Not a way to package or distribute application code

And, they likely require access to a modern API release to function as intended:

- *Kubernetes v1.19 or newer* is required for access to **`Pod Security Policies`** `[beta]`: https://k8s.io/docs/concepts/policy/pod-security-policy/
- *Kubernetes v1.16 or newer* required for **`admissionregistration.k8s.io/v1`**: https://k8s.io/docs/reference/access-authn-authz/extensible-admission-controllers/#prerequisites

Admission Control Basics

Part 1 Review:

- Understand how common Admission Controllers are enabled, disabled, and configured for a cluster
- Understand how configuration of Admission Controllers can be used to enforce basic security policies for a cluster
- Two phases, types of admission control: *Validating, Mutating*
- UX notes: It's a bit like a sledgehammer -
  Not designed to distribute quick/frequent policy changes
- #1 Use-case: Enforce consistent operational
  rules for disparate clusters in a release pipeline

Part 2
Dynamic Admission Control

Dynamic Admission Control

Requirements for Dynamic Admission Control:

1. **`DynMutatingAdmissionWebhook`**, and **`ValidatingAdmissionWebhook`** admission control plugins need to be enabled

2. When enabled, Kubernetes v1.16+ allows registration of Dynamic Admission Control webhooks via **`admissionregistration.k8s.io/v1`** via kind **`MutatingWebhookConfiguration`** and **`ValidatingWebhookConfiguration`**

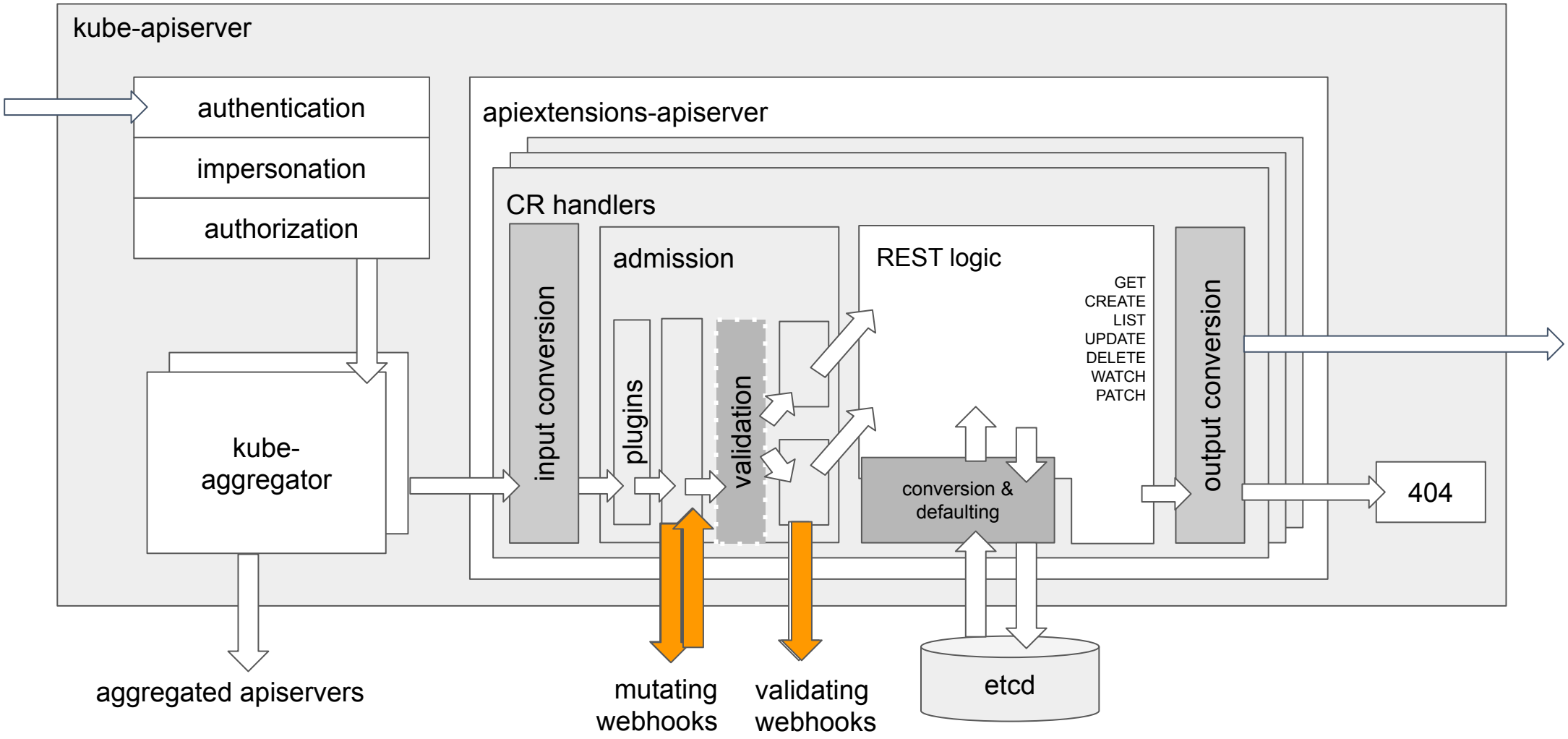
3. The name of a **`MutatingWebhookConfiguration`** or a **`ValidatingWebhookConfiguration`** object must be a valid DNS name

Docs:
https://k8s.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-configuration

Dynamic Admission Control


```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "pod-policy.example.com"
webhooks:
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  ["CREATE"]
    resources:   ["pods"]
    scope:       "Namespaced"
  clientConfig:
    service:
      namespace: "example-namespace"
      name: "example-service"
    caBundle: "Ci0tLS0tQk...<`caBundle` is a PEM encoded CA bundle which will be used to validate
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

Dynamic Admission Control



Dynamic Admission Control

Dynamic input validation use-cases:

- Always reject images tagged with ":latest"
- Require etcd member count to be an odd number between 1 and 11

GoClient Admission Webhook example:

https://k8s.io/docs/reference/access-authn-authz/extensible-admission-controllers/#write-an-admission-webhook-server

It is possible to implement an Admission Control Webhook server using *any language*, just make sure to return a proper response before "`timeoutSeconds`"

Dynamic Admission Control

Part 2 Review:

- Understand how Dynamic Admission Control webhooks are used to validate or coerce write requests as they pass through the API pipeline
  - The “Validate” phase will not begin until the Mutate phase has concluded (when Mutate is available)
- Review security use cases and implications

Part 3

When & How to avoid Admission Controllers

Alternatives

Select the appropriate abstraction for your scope of work:

Security Checklist:			Admin Required	Updates & Config
1. Establish an operational baseline for the cluster	→	Admission Control	Plugins	Dynamic
2. Establish operational rules for platform services and CRs	→	Operators	CRDs	Custom Resources
3. Establish application controls for development productivity	→	App CRs, Helm Charts		Chart Validation

☆ Then, establish mechanisms that ensure operational consistency throughout your release pipeline

Alternatives

Q: Why should I learn about Admission Controllers?

A: To establish strong security controls for shared-use clusters,
OR to help users find and adopt the appropriate tools for their scope of work

Q: Should I avoid writing (and maintaining) custom controllers that impact the operational reliability of the core platform APIs?

A: ***Yes*** - *Whenever possible!*

Q: Should I use Helm or Application CRDs if they are a viable option for my scope of work?

A: ***Yes*** - *Whenever possible!*

Alternatives

Is Helm is an option for you?

Use it!

Alternatives

Alternatives to Dynamic Admission Control:

Kubebuilder: `github.com/kubernetes-sigs/kubebuilder`

- Go

Operator Framework: `operatorframework.io`

- Go, Ansible, Helm

☆ OpenAPI Spec can provide schema-based input validation in the API pipeline (without introducing a Dynamic Admission Control webhook)

OpenAPI spec

Example code: OpenAPI spec schema validation

https://github.com/jdob/visitors-operator/blob/master/deploy/crds/example_v1_visitorsapp_crd.yaml

```
15    validation:
16      openAPIV3Schema:
17        properties:
18          apiVersion:
19            description: 'APIVersion defines the versioned
20              of an object. Servers should convert recogniz
21              internal value, and may reject unrecognized v
22            type: string
23          kind:
24            description: 'Kind is a string value representi
25              object represents. Servers may infer this fro
26              submits requests to. Cannot be updated. In Ca
27            type: string
```

Alternatives

Part 3 Review:

- Application CRs and Helm charts offer control interfaces that do not introduce the need for admin access privs (during normal operation)
- Consider using Kubebuilder or Operator SDK to provide validation and/or translation of API requests when it's an option for you
- Use OpenAPI Spec to provide schema-based input validation for Custom Resources

Review

Motivation:

- A reliable distributed platform that allows me to **focus on my day job**

Goals for this talk:

1. Understand the primary role of Admission Controllers
2. Understand typical use cases for Admission Control, and when to avoid this topic

Review

1. Primary Role of Admission Control:

- ★ API security controls (for platform admins)
- ★ Kubernetes API Request Translator
- ★ Kubernetes API Request Validator

Review

2. Use Cases for Admission Control:

Security:

- Deny “privileged” containers
- Deny escalation via abuse of hostPath, hostPID

Dynamic input validation:

- Always reject images tagged with “:latest”
- Require etcd members to be an odd number (between 1 and 11)

Purpose:

- Ensure operational consistency by enforcing basic security and policy for your platform
  - ✯ Standardize policy between clusters in a pipeline

Review

3. Alternatives to DIY Admission Control:

- App developers should stick with Helm or Application CRDs, avoid making unnecessary modifications to the API pipeline
- Consider using Kubebuilder or Operator SDK for dynamic validation and/or translation of API requests
- Use OpenAPI Spec to provide schema-based input validation
- Compare Kubernetes hosting providers (and/or distributions) that include a strong set of Admission Control defaults, and a clear plan for distributing updates
  - http://learn.openshift.com (1 hour session)
  - http://openshift.com/try

Links & Resources

Admission Control:

http://k8s.io/docs/reference/access-authn-authz/admission-controllers

Dynamic Admission Control:

https://k8s.io/docs/reference/access-authn-authz/extensible-admission-controllers/

Vulnerability Disclosures: **https://k8s.io/security**

Recommended Talks:

- The Path Less Traveled: Abusing Kubernetes Defaults
  https://youtu.be/gtaaONq-XGY
- Customizing and Extending the Kubernetes API with Admission Controllers
  https://youtu.be/P7QAfjdbogY
- Admission Webhooks: Configuration and Debugging Best Practices
  https://youtu.be/r_v07P8Go6w
- Deep Dive: API Machinery SIG
  https://youtu.be/kz8BMn9_hk8

Q & A

Thank You!

@RyanJ at Red Hat

bit.ly/k20ac
sched.co/ekBb

Admission Control, We Have a Problem

Ryan Jarvinen, Red Hat

bit.ly/k20ac
sched.co/ekBb


KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual