Managing the Kubernetes API Server and Pods

INTRODUCTION AND USING THE KUBERNETES API

Kien Bui
DevOps & Platform Engineer

Course Overview

Using the Kubernetes API

Managing Objects with Labels, Annotations, and Namespaces

Running and Managing Pods

Overview

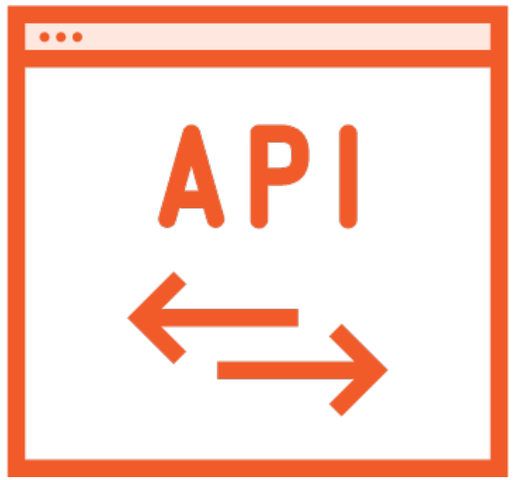
The Kubernetes API and API Server

Working with Kubernetes Objects

- Defining objects
- API Groups
- API Versioning

Anatomy of an API Request

Kubernetes API and API Server



Single surface area over the resources in your data center

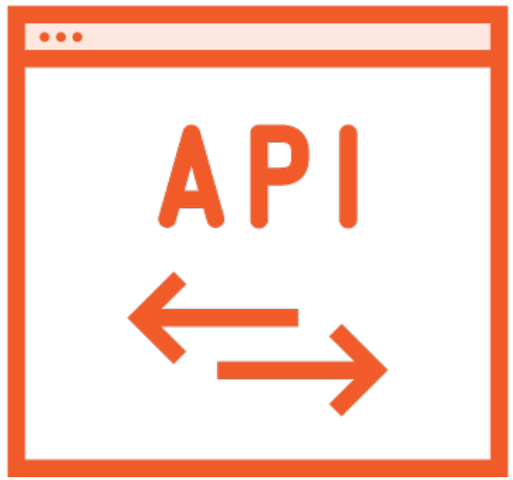
API Objects

- Collection of primitives to represent your system's state
- Enables configuration of state

API Server

- The sole way to interact with your cluster
- The sole way Kubernetes interacts with your cluster

Kubernetes API Server



Client/Server architecture

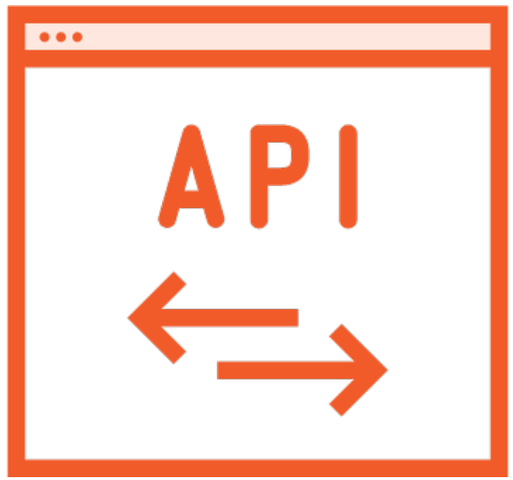
- RESTful API over HTTP using JSON
- Client submits requests over HTTP/HTTPS
- Server responds to the request

Stateless

Serialized and persisted in the cluster store

Control Plane Node

Kubernetes API Objects



Persistent entities in Kubernetes

Representing the state of your system

Objects are organized by

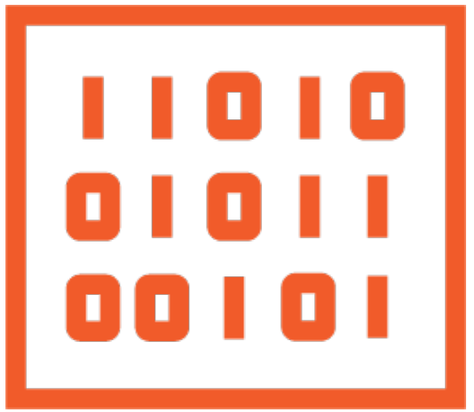
- Kind - `Pod`, `Service`, `Deployment`
- Group - core, apps, storage
- Version - **`v1`**, **`beta`**, **`alpha`**

Kubernetes API Objects (Kind)

Pods

Deployments

Services



PersistentVolumes

Not an exhaustive list, but these are the key players

Working with Kubernetes Objects

Imperative configuration

Declarative configuration

- Define our desired state in code
- Manifest
- YAML or JSON

```
kubectl apply -f deployment.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx
```

Basic Manifest - Pod

```
kubectl apply -f nginx.yaml
```

https://kubernetes.io/docs/reference/kubernetes-api/

Working with kubectl dry-run

Server-side

- Processed as a typical request
- Requests will NOT be persisted in storage

Client-side

- Writes the object to be created to `stdout`
- Validate manifest syntax
- Great for generating syntactically correct YAML manifests

Using kubectl dry-run

```
kubectl apply -f deployment.yaml --dry-run=server

kubectl apply -f deployment.yaml --dry-run=client

kubectl create deployment nginx --image=nginx \
  --dry-run=client -o yaml

kubectl create deployment nginx --image=nginx \
  --dry-run=client -o yaml > deployment.new.yaml
```

Working with kubectl diff

Generates the difference between

- Resources running in the cluster
- Resources defined in a manifest or `stdin`

Outputs the differences to `stdout`

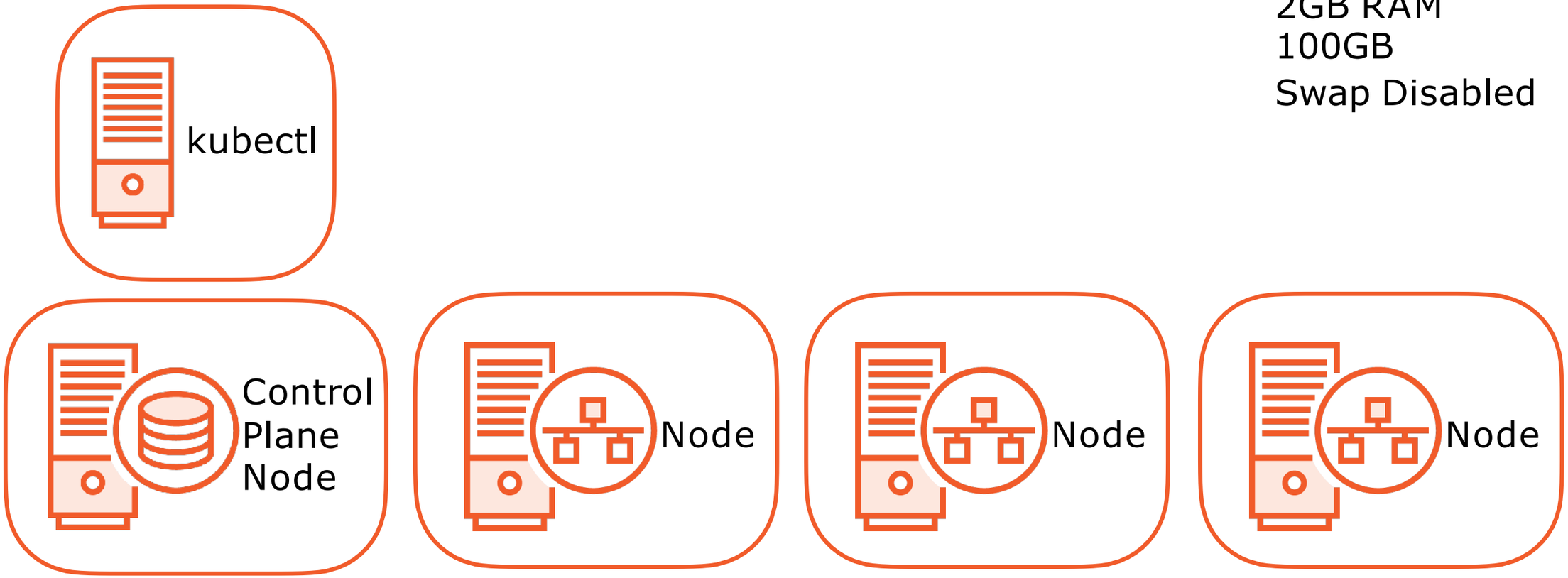
Useful to help you understand what's going to change

```
kubectl diff -f newdeployment.yaml
```

Hostnames set
Host file on each

Lab Environment

Ubuntu 18.0.4
VMware Fusion VMs
2vCPU
2GB RAM
100GB
Swap Disabled



c1-cp1
172.16.94.10

c1-node1
172.16.94.11

c1-node2
172.16.94.12

c1-node3
172.16.94.13

Demo

API Server Discovery

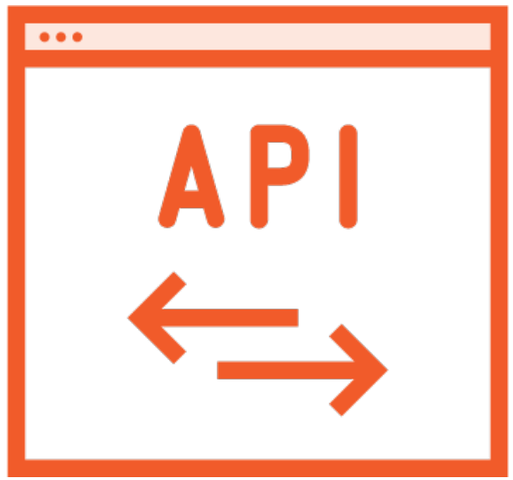
Listing Available API Resources

Using `kubectl explain`

Defining objects in YAML

Working with `kubectl dry-run` and `diff`

API Groups



Organization of resources

API Groups

- Core API (Legacy Group)
- Named API Groups

Part of the API Object's URL in API Requests

API Groups

Core (Legacy)

Pod

Node

Namespace

PersistentVolume

PersistentVolumeClaim

Named API Groups

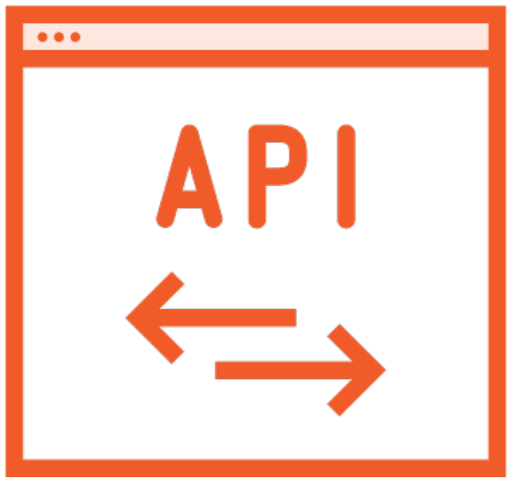
`apps` - **Deployment**

`storage.k8s.io` - **StorageClass**

`rbac.authorization.k8s.io` - **Role**

https://kubernetes.io/docs/reference/kubernetes-api/

API Versioning



API is versioned

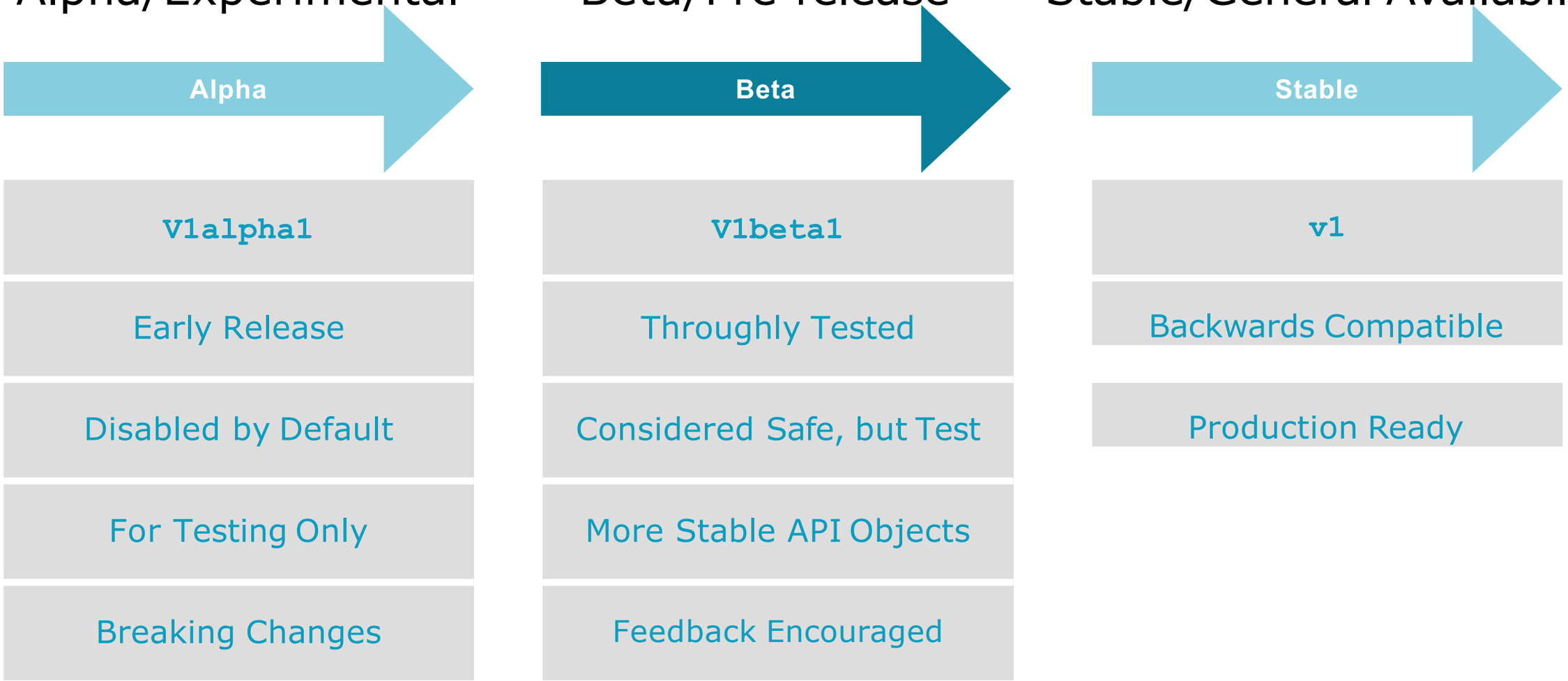
Provide stability for existing implementations

Enable forward change

Alpha -> Beta -> Stable

No direct relation to release versions

API Versioning

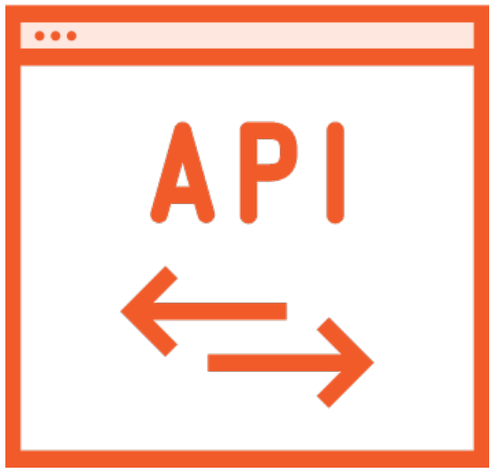


Demo

API Object Discovery

- Examining API Groups
- Examining specific API Versions

Anatomy of an API Request



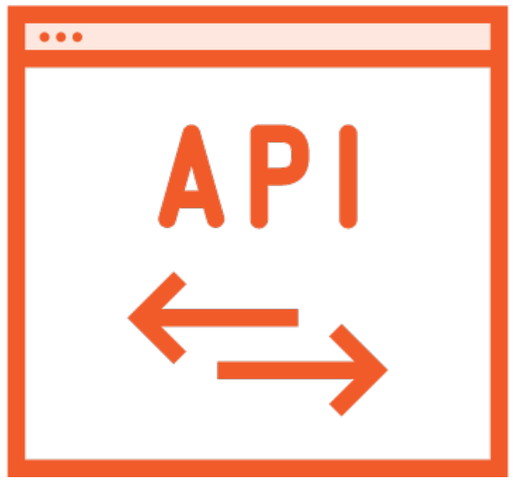
API Request

API Paths

Read/Write Objects to/from Cluster Store

Send Response Back to the Client

Anatomy of an API Request



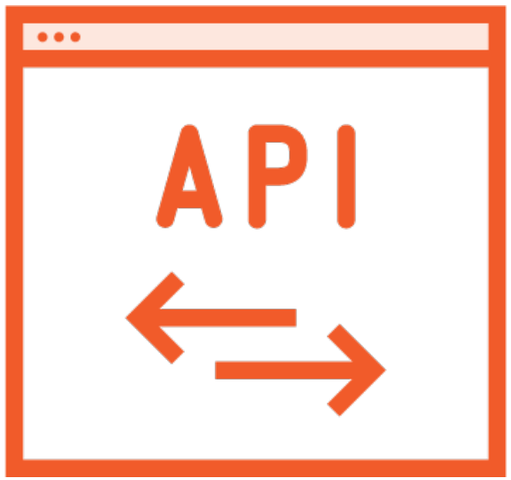
Client and Server architecture

`kubectl`

Any HTTP client that respects the API

`curl`

Anatomy of an API Request



HTTP based RESTful API

- HTTP Verb
- Resource Location (URL/Path)
- Request = Verb + Resource Location
- Response Code

RESTful API Verbs

GET	Get the data for a specified resource(s)
POST	Create a resource
DELETE	Delete a resource
PUT	Create or update entire existing resource
PATCH	Modify the specified fields of a resource

Special API Requests

LOG	Retrieve logs from a container in a Pod
EXEC	Exec a command in a container get the output
WATCH	Change notifications on a resource with streaming output

Each resource has a resourceVersion

Watches are started on that version

Notifications are sent to clients watching that version

API Resource Location (API Paths)

Core API (Legacy)

```
http://apiserver:port/api/$VERSION/$RESOURCE_TYPE

http://apiserver:port/api/$VERSION/namespaces/$NAMESPACE/$RESOURCE_TYPE/$RESOURCE_NAME
```

API Groups

```
http://apiserver:port/apis/$GROUPNAME/$VERSION/$RESOURCE_TYPE

http://apiserver:port/apis/$GROUPNAME/$VERSION/namespaces/$NAMESPACE/$RESOURCE_TYPE/$RESOURCE_NAME
```

Response Codes from the API Server

Success (2xx)	Client Errors (4xx)	Server Errors (5xx)
200 - OK	401 - Unauthorized	500 - Internal Server Error
201 - Created	403 - Access Denied	
202 - Accepted	404 - Not Found	

Anatomy of an API Request

Client Request

Server Response

Can you make a connection?	Are you valid user?	Can you perform the requested action?	Administrative control over request
HTTP over TCP	Authentication plugin	Verb on Resource	Additional code
TLS Encrypted	Modular	Default deny	May modify object
	401	403	Validation

Demo

Anatomy of an API Request

Special API Requests - Watch, Exec and Log

Authentication Failure and Missing Resources

Creating Objects

Summary

The Kubernetes API and API Server

Working with Kubernetes Objects

- Defining objects
- API Groups
- API Versioning

Anatomy of an API Request

What’s Next!

Managing Objects with Labels, Annotations, and Namespaces