Extending  Kubernetes

Radu Munteanu – 1 November 2018

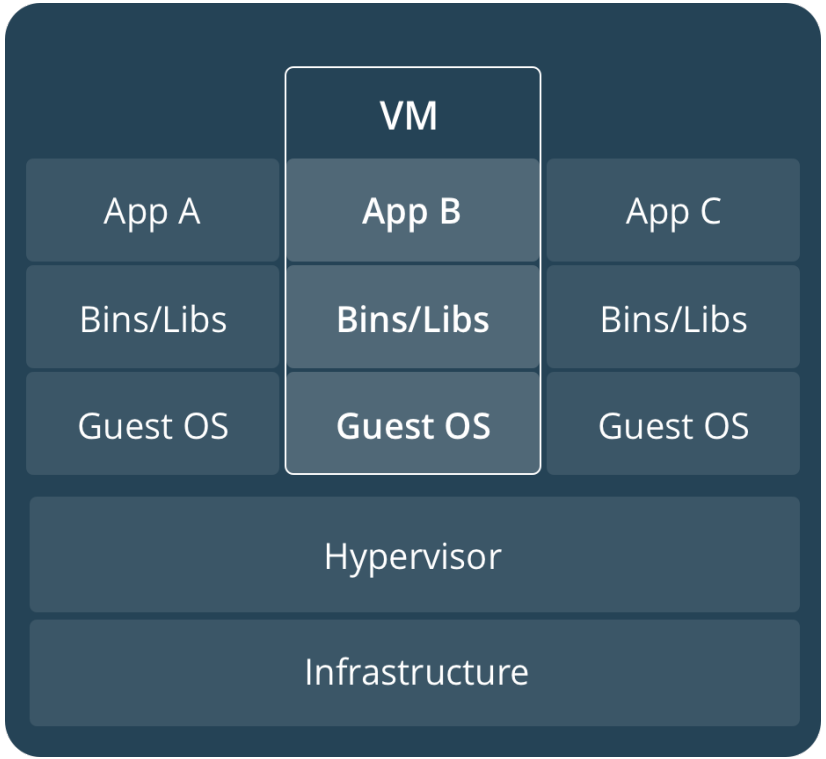
DevOps Meetup / Cloud Native Computing Iași

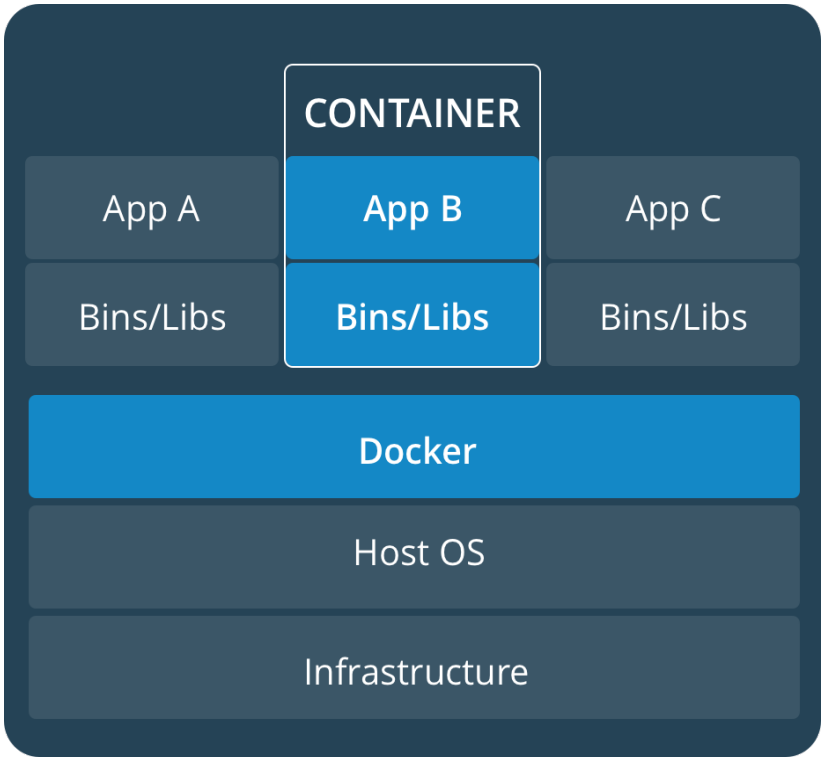
Contents

- Why Kubernetes
- Why Extend
- Kubernetes Architecture
- What to Extend
- How to Extend
- Demo

Why Kubernetes

- Virtual Machines vs Containers (docker)

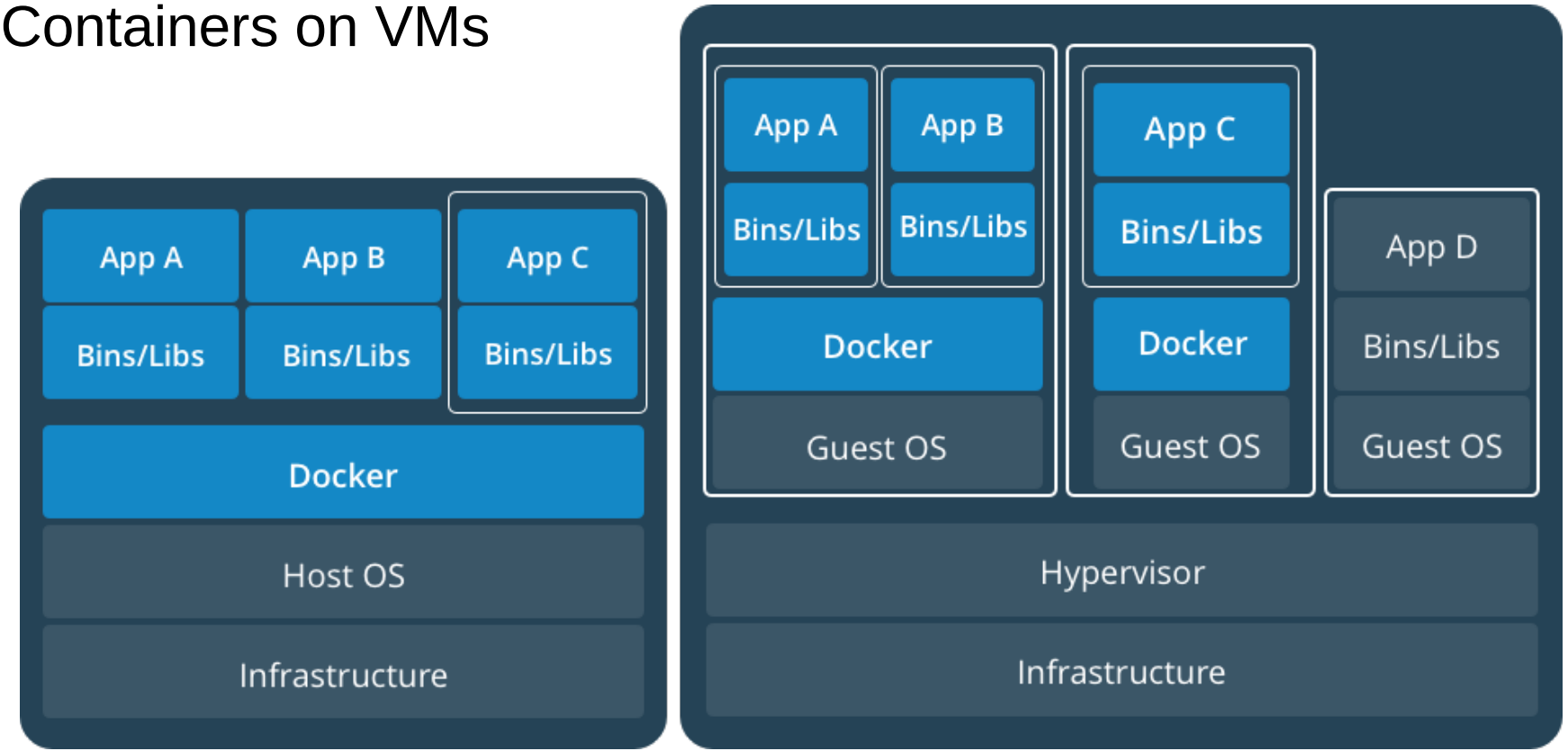




https://www.docker.com/what-container

Why Kubernetes

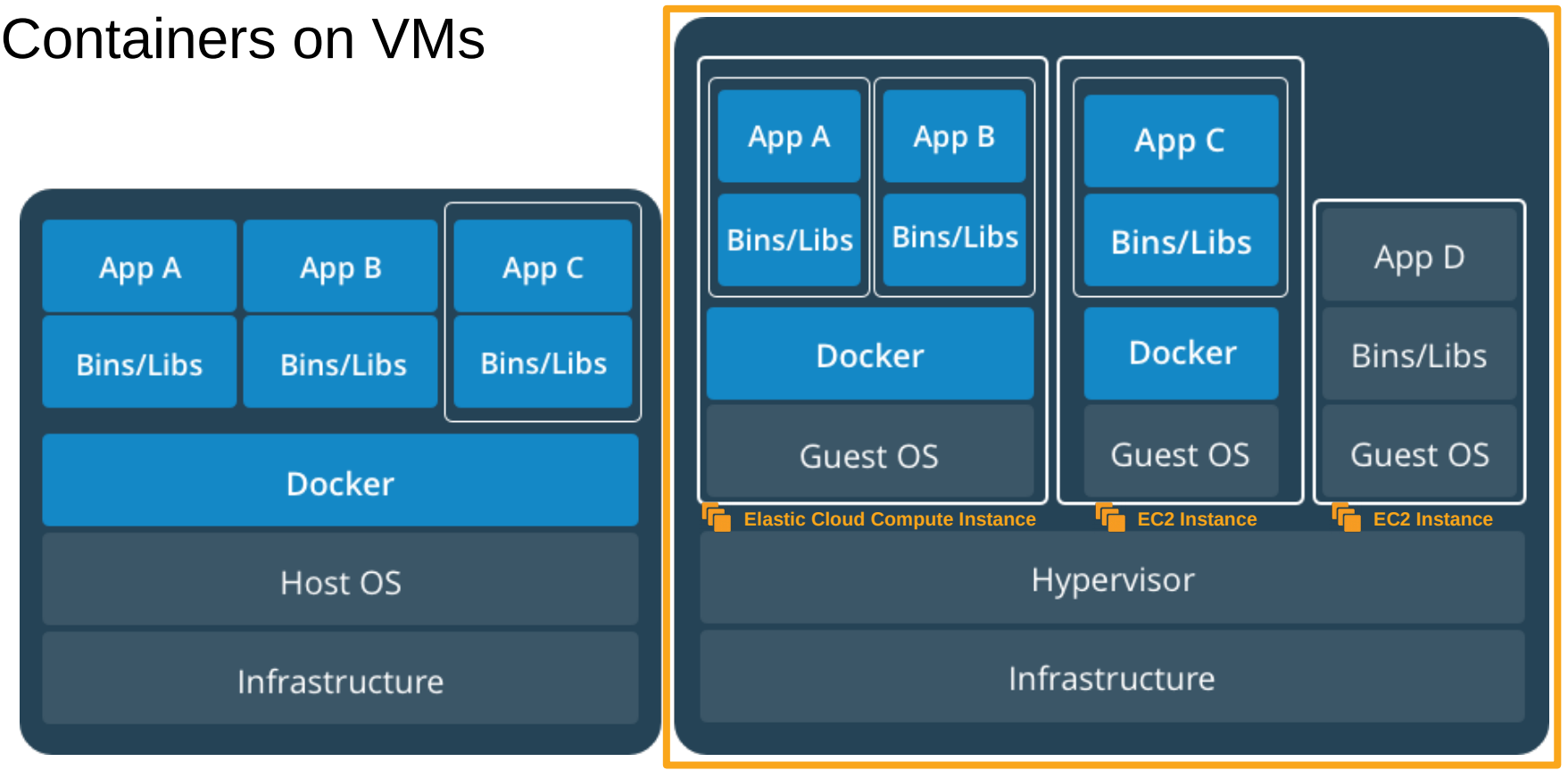
- Containers on VMs



https://www.docker.com/what-container

Why Kubernetes

- Containers on VMs



https://www.docker.com/what-container

Why Kubernetes

-  Kubernetes (K8s)
  - Container communication over multiple hosts
  - Higher level of abstraction: resources

    (E.g. nodes, pods, replica sets, deployments, services)

    => Easier definition of virtual resources (VMs, apps, proxies)

    => Automatic management of failures (replica sets, deployments)
  - Central management of containers (Dashboard or *kubectl*)
  - One declarative, domain-specific language for managing all resources within the Kubernetes cluster (common format: *YAML*)

Why Extend

- Easier to manage the whole infrastructure using the same language and tools
  - Managing AWS resources
    - Now: AWS Console → Cloud Formation DSL, aws-cli, aws-api
    - Future: Cloud Formation DSL, aws-cli, aws-api → K8s DSL, kubectl
  - Managing installation and deployments
    - aws-cli, scripts → K8s DSL, kubectl
- Standardize communication and configuration across all apps using the features and extensibility of K8s
  - Service Discovery: Zookeeper → K8s Service Catalog
  - Configuration: Zookeeper → Custom Resources, Custom K8s API

Kubernetes Architecture

-  Resources (Objects *)
  - A Kubernetes resource is a declarative API with a well defined **Schema** structure and **endpoints**. Because the structure of the Schema and Endpoints are both well understood, many Kubernetes tools support all APIs written as Kubernetes resources
  - **Schema: Group, Version, Kind**

    Every Kubernetes resource has a Group, Version and Kind that uniquely identifies it
    - The resource **Kind** is the name of the API - such as Deployment or Service
    - The resource **Version** defines the stability of the API and backward compatibility guarantees - such as v1beta1 or v1
    - The resource **Group** is similar to package in a language. It disambiguates different APIs that may happen to have identically named Kinds. Groups often contain a domain name, such as k8s.io

```
apiVersion: apps/v1
kind: Deployment
```

```
apiVersion: radu/v1alpha1
kind: MyResource
```

* https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects

http://book.kubebuilder.io/basics/what_is_a_resource.html

Kubernetes Architecture

-  Resources
  - **Schema: Spec, Status, Metadata**

    Most Kubernetes resources Schema contain 3 components: Spec, Status and Metadata
    - **Spec**: defines the desired state of the cluster as specified by the user
    - **Status**: publishes the state of the cluster as observed by the controller
    - **Metadata**: contains information common to most resources about the object including as the object name, annotations, labels and more

http://book.kubebuilder.io/basics/what_is_a_resource.html

Kubernetes Architecture

- Resources

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 1
  template:
    spec:
      containers:
      - image: nginx
        name: nginx
status:
  replicas: 1
  unavailableReplicas: 1
  updatedReplicas: 1
```

http://book.kubebuilder.io/basics/what_is_a_resource.html

Kubernetes Architecture

-  Resources
  - **Endpoints**

    Kubernetes resources have well defined endpoints as described below
    - **Create, Update, Patch, Delete**

      The create, update, patch and delete endpoints may be used to modify objects. The update endpoint replaces the object with what is provided, whereas the patch endpoint selectively updates fields
    - **Get, List, Watch**

      The get, list and watch endpoints may be used to get a specific resource by name, list all resources matching a labels, or continually watch for updates

http://book.kubebuilder.io/basics/what_is_a_resource.html

Kubernetes Architecture

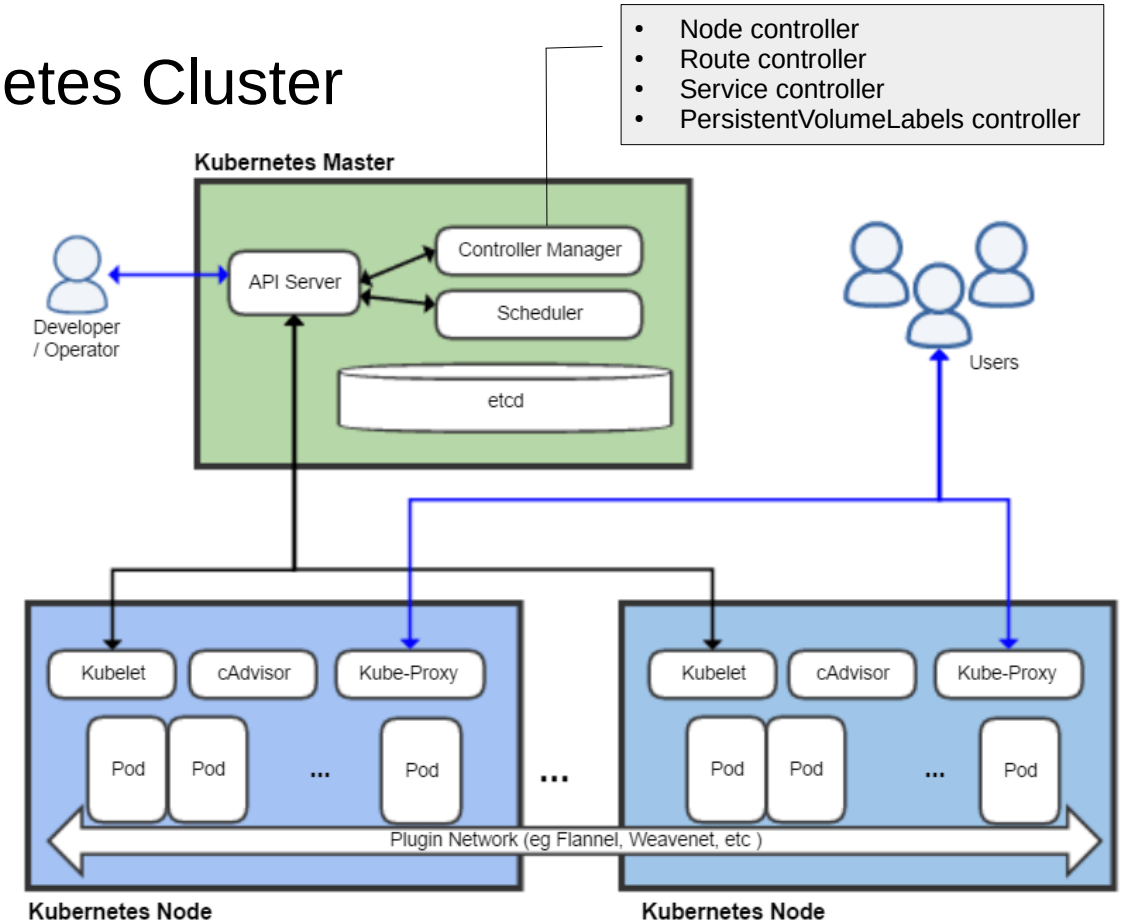
- Namespaces
  - Kubernetes supports multiple virtual clusters backed by the same physical cluster. These **virtual clusters** are called namespaces
    - Namespaces are intended for use in environments with many users spread across multiple teams, or projects. For clusters with a few to tens of users, you should not need to create or think about namespaces at all. Start using namespaces when you need the features they provide
    - Namespaces provide a **scope** for names. **Names of resources need to be unique within a namespace, but not across namespaces**
    - Namespaces are a way to divide cluster resources between multiple users (via resource quota)

```
$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    1d
kube-system   Active    1d
kube-public   Active    1d
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces

Kubernetes Architecture

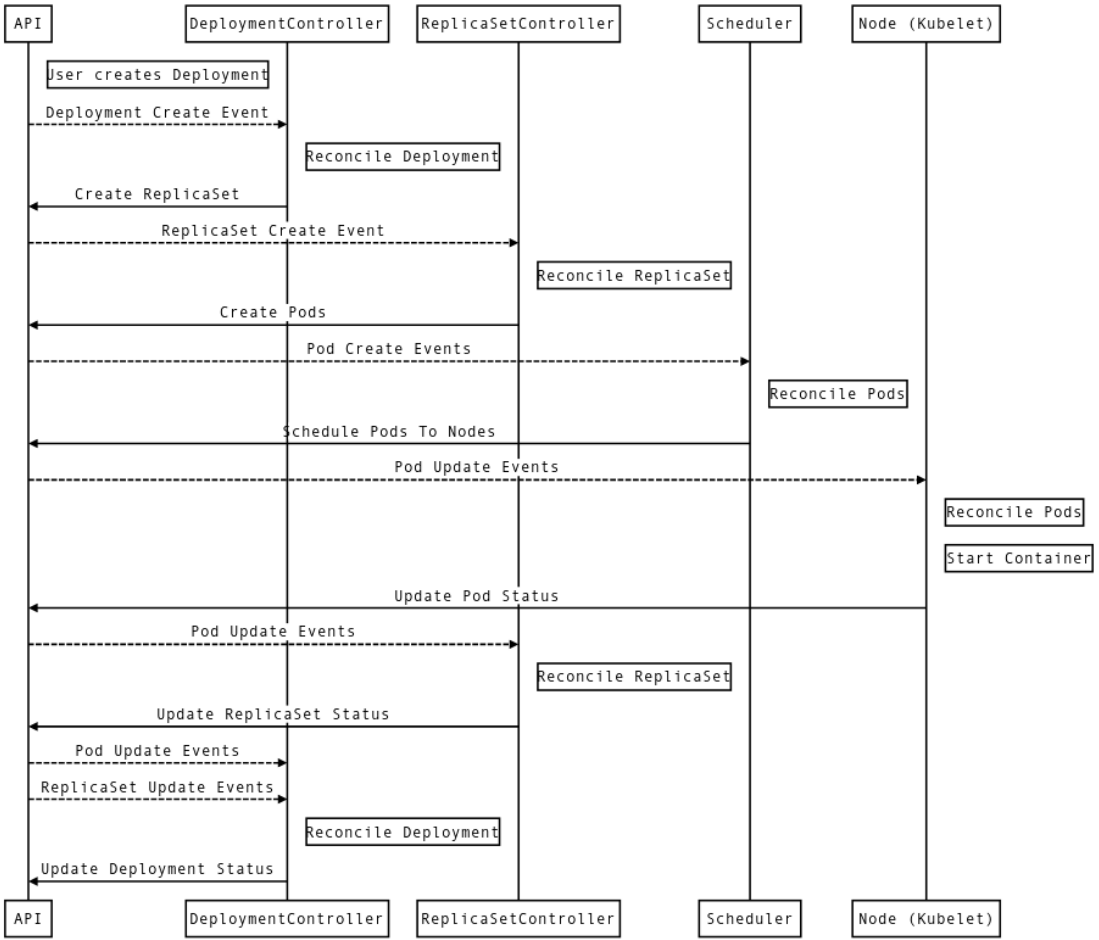
- Kubernetes Cluster



https://en.wikipedia.org/wiki/Kubernetes

Kubernetes Architecture

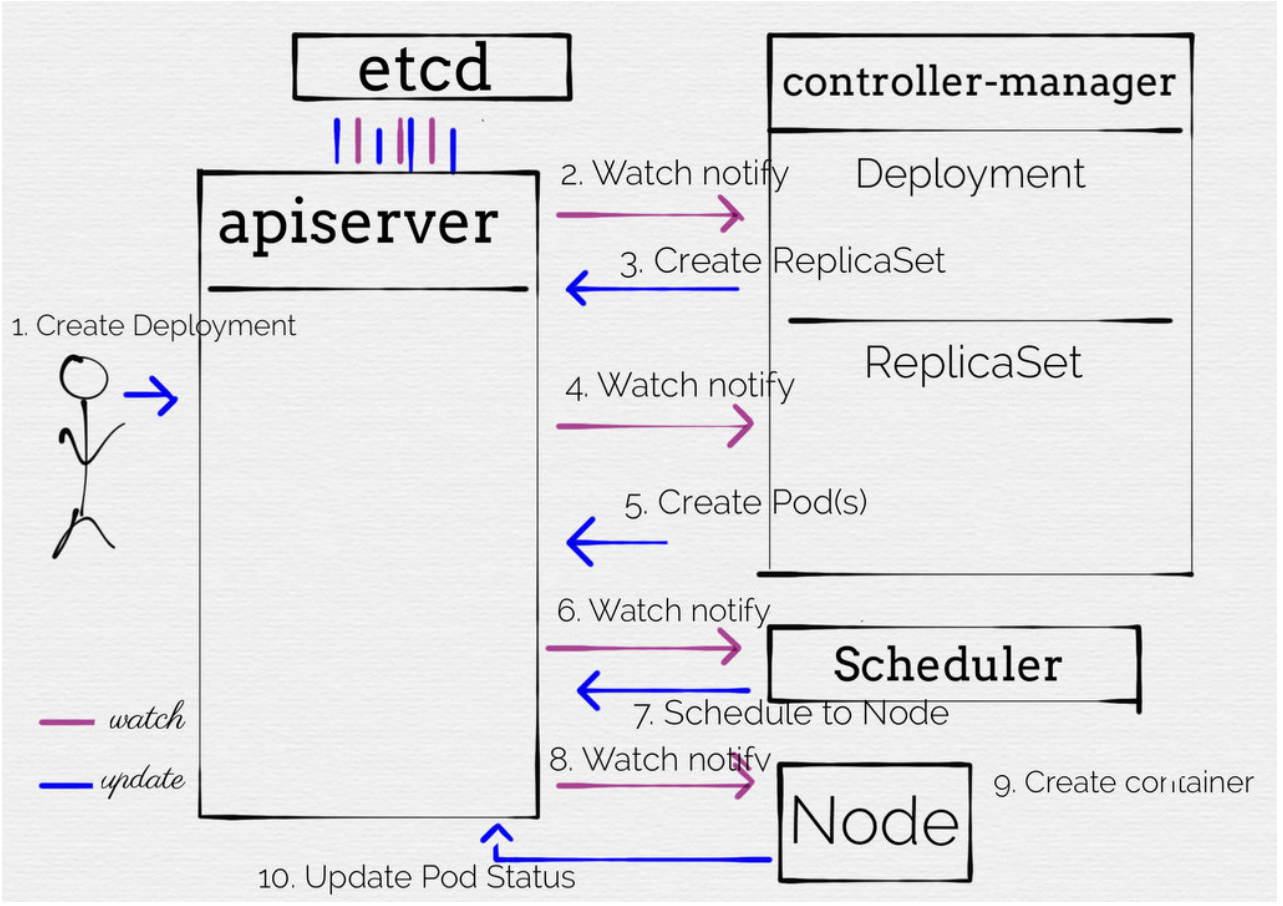
- Flow



http://book.kubebuilder.io/basics/what_is_a_controller.html

Kubernetes Architecture

- Flow



https://github.com/kubernetes-incubator/apiserver-builder/blob/master/docs/concepts/api_building_overview.md

What to Extend

- **Kubectl Plugins**
- Service Catalog
- **Custom Resources**
- Custom Schedulers
- **Custom Controllers**
- Custom APIs
- Network Plugins
- Storage Plugins
- ...

https://kubernetes.io/docs/concepts/extend-kubernetes

How to Extend – Kubectl Plugins

- Calling a Plugin

```
kubectl plugin <plugin_name> [flags] [options]
```

- Plugin Structure

```
~/.kube/plugins/
└── targaryen
    ├── plugin.yaml
    └── dracarys
```

- Plugin YAML

```
name: "targaryen"                  # REQUIRED: the plugin command name, to be invoked under 'kubectl'
shortDesc: "Dragonized plugin"     # REQUIRED: the command short description, for help
longDesc: ""                       # the command long description, for help
example: ""                        # command example(s), for help
command: "./dracarys"              # REQUIRED: the command, binary, or script to invoke
flags:                             # flags supported by the plugin
  - name: "heat"                   # REQUIRED for each flag: flag name
    shorthand: "h"                 # short version of the flag name
    desc: "Fire heat"              # REQUIRED for each flag: flag description
    defValue: "extreme"            # default value of the flag
tree:                              # allows the declaration of subcommands
  - ...                            # subcommands support the same set of attributes
```

- Environment Variables

```
KUBECTL_PLUGINS_
```


https://v1-11.docs.kubernetes.io/docs/tasks/extend-kubectl/kubectl-plugins

How to Extend – Kubectl Plugins

- In V.1.12
- Calling a Plugin

```
kubectl <plugin_name> [flags] [options]
```

- Plugin Structure
  - No more structure, the plugin is any app found in **PATH** starting with **kubectl-**

    ```
    kubectl-<plugin_name>
    ```

  - Subcommands are defined by adding a dash in the name

    ```
    kubectl-<plugin_name>-<subcommand1>-<subcommand2>
    ```

    - E.g.: `kubectl foo bar baz` → `kubectl-foo-bar-baz`
  - Define a plugin with the name **hello-world**?

    ```
    kubectl-hello_world
    ```

    https://kubernetes.io/docs/tasks/extend-kubectl/kubectl-plugins

How to Extend – Custom Resources

- Custom Resource Definition (CRD)

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: crontabs.stable.example.com
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: stable.example.com
  # version name to use for REST API: /apis/<group>/<version>
  version: v1
  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: crontabs
    # singular name to be used as an alias on the CLI and for display
    singular: crontab
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: CronTab
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
    - ct
```

https://kubernetes.io/docs/tasks/access-kubernetes-api/extend-api-custom-resource-definitions

How to Extend – Custom Resources

- CRD

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: crontabs.stable.example.com
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: stable.example.com
  # list of versions supported by this CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: crontabs
    # singular name to be used as an alias on the CLI and for display
    singular: crontab
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: CronTab
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
    - ct
```

https://kubernetes.io/docs/tasks/access-kubernetes-api/extend-api-custom-resource-definitions

How to Extend – Custom Resources

- Why?
  - Automate your infrastructure through Kubernetes DSL
  - Interact with Operators* for your Cloud Native Apps

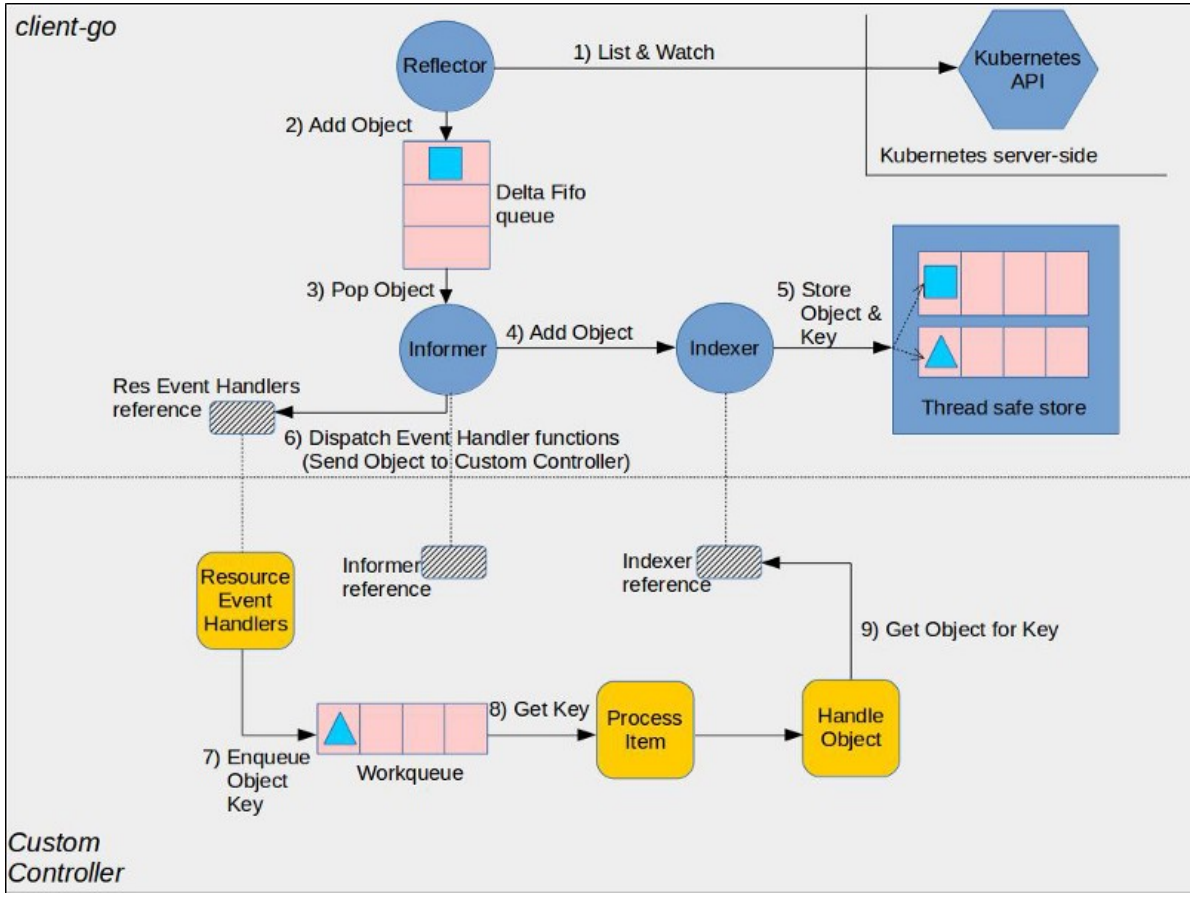
* “Some controllers are referred to as Operators. Operators are a specific type of controller that manage running a specific application such as Redis or Cassandra.” (http://book.kubebuilder.io/basics/what_is_a_controller.html)

How to Extend – Custom Resources

- How to use in Controllers?

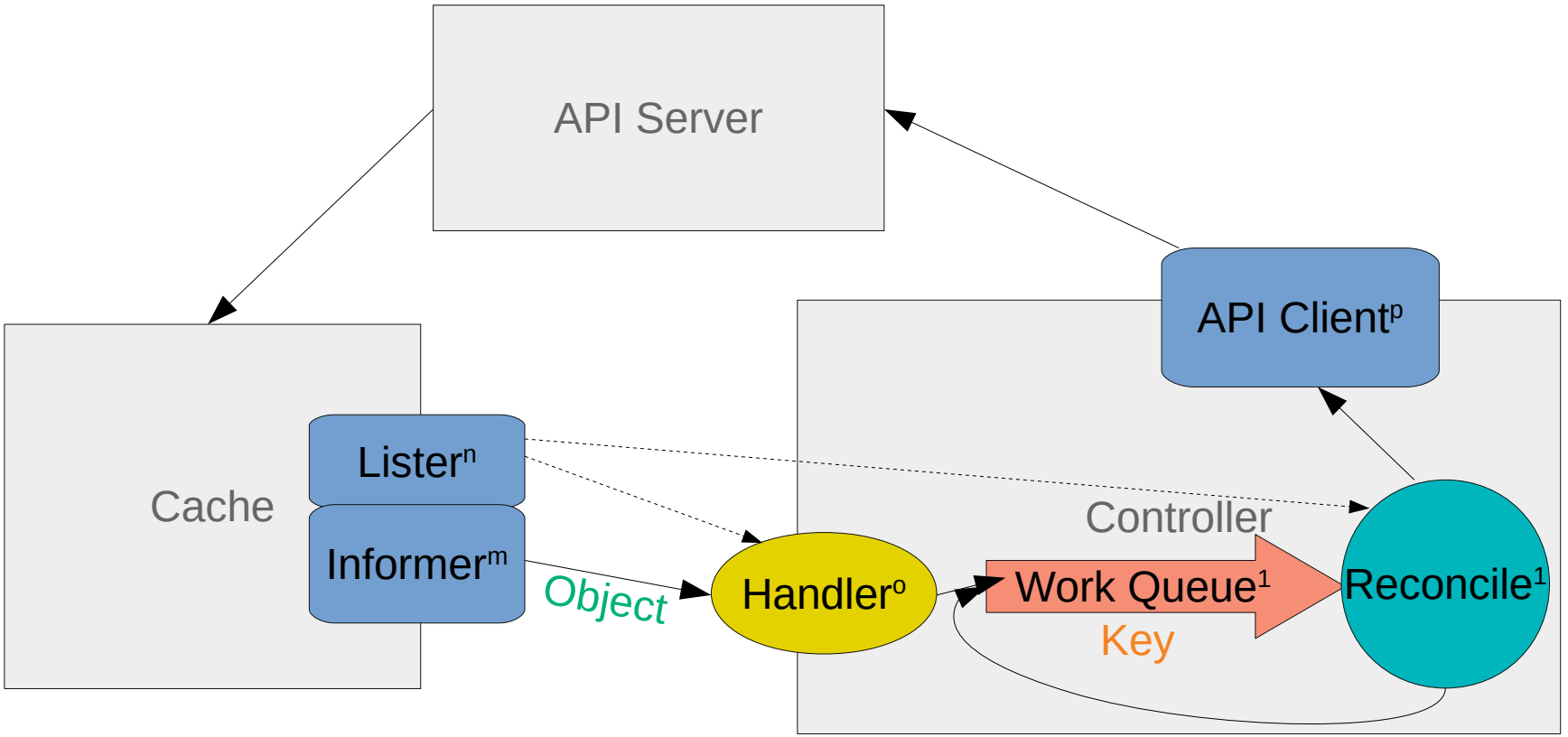
```
// Foo is a specification for a Foo resource
type Foo struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   FooSpec   `json:"spec"`
    Status FooStatus `json:"status"`
}
```

How to Extend – Custom Controllers



https://medium.com/@cloudark/kubernetes-custom-controllers-b6c7d0668fdf

How to Extend – Custom Controllers



How to Extend – Custom Controllers

- Leader Election
  - "k8s.io/client-go/tools/leaderelection"
    - https://github.com/kubernetes/client-go/tree/master/tools/leaderelection
  - Resource Locks:
    - Endpoints
    - ConfigMaps

How to Extend – Custom Controllers

- There’s no mature library or framework
- What can you use right now:
  - Modifying existing controllers (like sample-controller)
  - KubeBuilder
  - MetaController
  - Operator SDK
  - k8s.io Code-Generator & co (client-go, apimachinery, etc.)
  - sigs.k8s.io controller-runtime, controller-tools

https://kubernetes.io/docs/concepts/extend-kubernetes

Demo

- sample-controller
  - https://github.com/kubernetes/sample-controller
- k8s-sample-plugin
  - https://gitlab.com/radu-munteanu/k8s-sample-plugin

Q & A