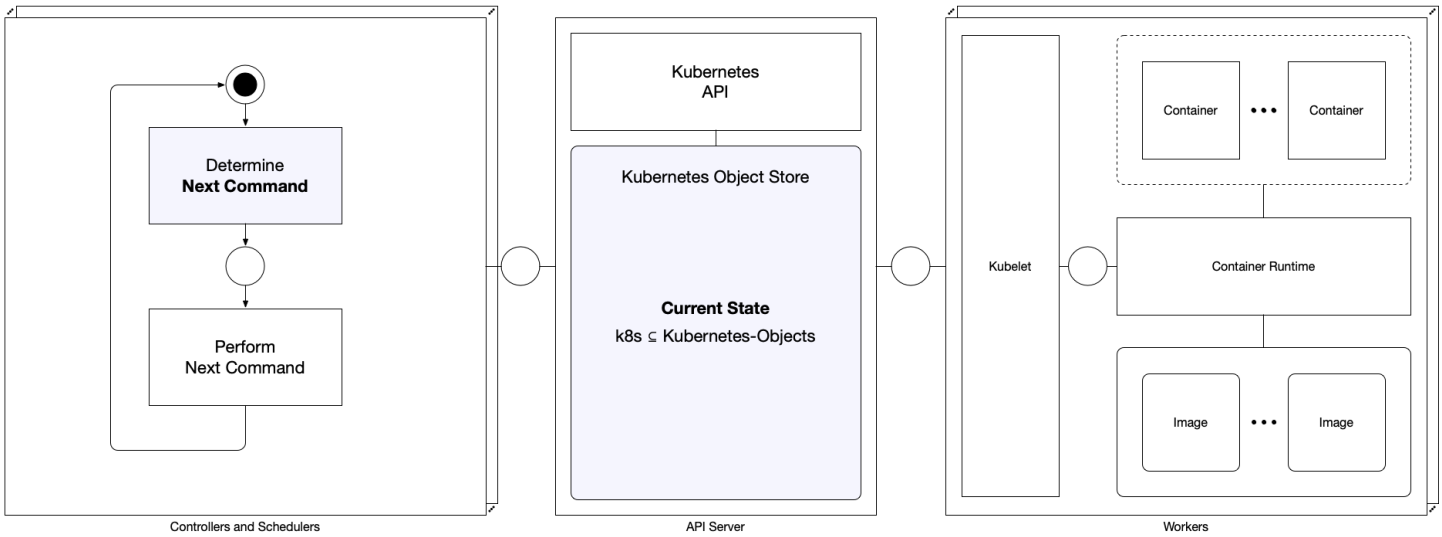


Kubernetes — Conceptual Architecture

The Mechanics of Kubernetes

Dominik Tornow
Nov 25, 2018 · 5 min read

By Andrew Chen and Dominik Tornow

Kubernetes is a Container Orchestration Engine designed to host containerized applications on a set of nodes, commonly referred to as a cluster. Using a systems modeling approach this series aims to advance the understanding of Kubernetes and its underlying concepts.

For this blog post, an advanced understanding of Kubernetes, Kubernetes Objects, and Kubernetes Controllers is recommended.

Kubernetes is characterized as a declarative Container Orchestration Engine: In a declarative system, the user supplies a representation of the desired state *of* the system *to* the system. Then, the system considers the *current state* and the *desired state* to determine the sequence of commands to transition from current state to desired state.

Therefore, the term “declarative system” sparks the notion of a calculated, coordinated effort with the explicit purpose to transition from the current state to the desired state.

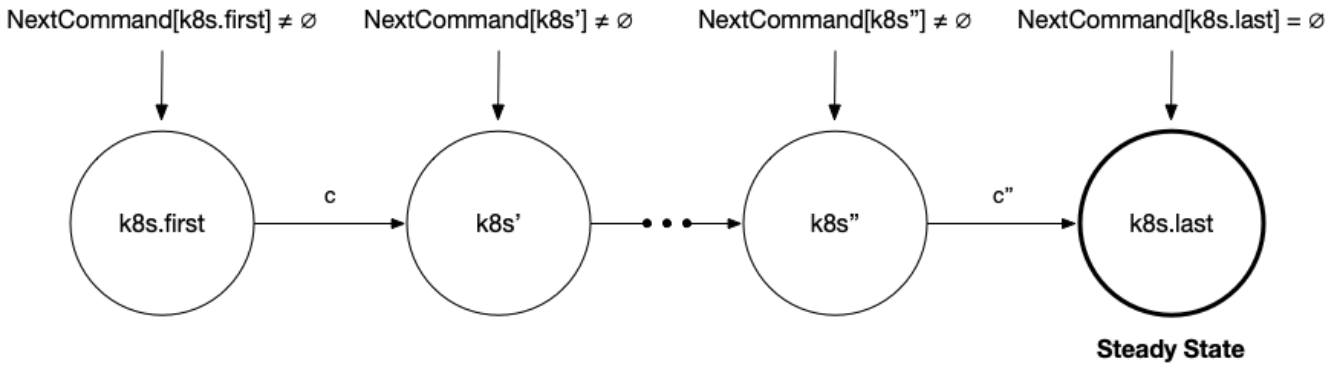
However, this is **not** how Kubernetes **actually** works!

Kubernetes does not determine a calculated, coordinated sequence of commands to execute based on the current state and the desired state.

Instead, Kubernetes iteratively determines the next command to execute based on the current state only. If and when no next command can be determined, Kubernetes reached a *steady state*.

State Transition Mechanics

This paragraph outlines an abstract model of Kubernetes' state transition semantics. The next paragraphs outline a concrete example based on Deployment Objects and the Deployment Controller.




```
fact {
    all k8s : K8s - last | let k8s' = k8s.next {
        some c : NextCommand[k8s] {
            command.source = k8s and command.target = k8s'
        }
    }
    NextCommand[k8s.last] = none
}
```

k8s-mechanics-kubernetes.als hosted with ❤ by **GitHub** **view raw**

Listing 1. State Transition Mechanics of Kubernetes

Listing 1. specifies the state transition semantics of Kubernetes: Given a Next Command Function, the system will determine the next command based on the current state, k8s, that transitions the system from its current state, k8s, to its next state, k8s'.

```
fun NextCommand(k8s : K8s) : set Command {
  DeploymentController.NextCommand[k8s] +
  ReplicaSetController.NextCommand[k8s] +
```

```
    ...
}
```

k8s-mechanics-next-command.als hosted with ❤ by GitHub view raw

Listing 2. Next Command Function

Conceptually, the Next Command Function is the composition of the Next Command Functions of every Kubernetes Controller.

```
pred Steady(k8s : K8s) { NextCommand[k8s] = none }
```

k8s-mechanics-steady.als hosted with ❤ by GitHub view raw

Listing 3. Steady State Predicate

The sequence of states is terminated by a state k8s.last for which the Next Command function does not yield a next command, a state commonly referred to as Steady State.

Enter Kubernetes Objects

The Kubernetes Object Store is a set of Kubernetes Objects. Kubernetes Objects are data records that come in different flavors, called kinds.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: busybox
        image: busybox
```

k8s-mechanics-deployment-object.yml hosted with ❤ by GitHub view raw

Listing 4. Deployment Object — Record-Of-Fact or Record-Of-Intent?

Listing 4. illustrates a Deployment Object: The Listing states the *fact* that there *exists* a Kubernetes Object, so that the object's

- `.kind` equals `Deployment`
- `.spec.replicas` equals `3`

- `.spec.template.spec.containers[0].image` equals `BusyBox`

Enter Kubernetes Controllers

Every Kubernetes Controller contributes to the Next Command function: A Controller is implemented as a continuous process that yields commands based on Kubernetes' current state.

```
process Controller = "Deployment Controller"
begin
    ControlLoop:
      while TRUE do
        \* The Deployment Controller monitors Deployment Objects
        with d ∈ {d ∈ k8s: d.kind = "Deployment"} do
          \* 1. Enabling Condition
          if Cardinality({r \in k8s: r.kind = "ReplicaSet" ∧ match(d.spec.labelSelecto
            \* Reconciling Command
            CREATE([kind |-> "ReplicaSet", spec |-> [replicas |-> d.spec.replicas, temp
          end if;
          \* 2. Enabling Condition
          if Cardinality({r \in k8s: r.kind = "ReplicaSet" ∧ match(d.spec.labelSelecto
            \* Reconciling Command
            with r ∈ {r \in k8s: r.kind = "ReplicaSet" ∧ match(d.spec.labelSelector, r
               DELETE(r);
            end with;
          end if;
        end with;
      end while;
end process;
```

k8s-mechanics-deployment-controller.tla hosted with ♥ by GitHub view raw

Listing 5. (Simplified) Deployment Controller

Listing 5. illustrates the the Deployment Controller: The Controller monitors Deployment Objects and for each Object performs a set of conditional statements:

- **Condition**
  If there is less than 1 matching ReplicaSet Object
  Command
  then the Deployment Controller will yield a Create ReplicaSet Command

- **Condition**
  If there is more than 1 matching ReplicaSet Object

Command

then the Deployment Controller will issue a Delete ReplicaSet Command

From the point of view of a Controller, Kubernetes is in *steady state* if none of the Controller's conditions are enabled, that is, the Controller will not yield any command.

Cascading Commands

Controllers (may) cascadingly enable each other:

- Given the state k8s, if a Kubernetes Controller C is enabled, C will perform a command that transitions to k8s'.
- Given the state k8s', if Kubernetes Controller C' is enabled, C' will perform a command that transitions to k8s"

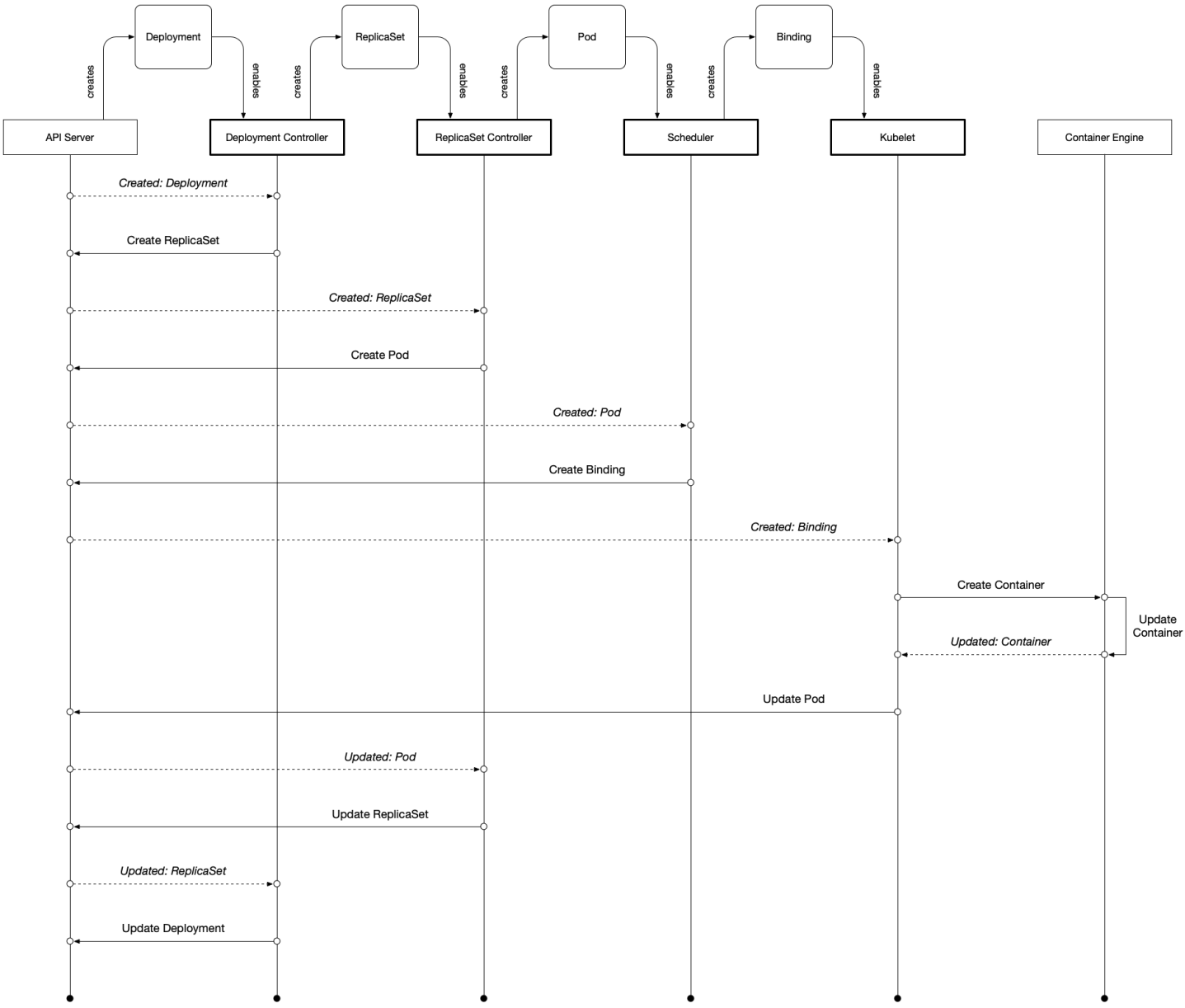
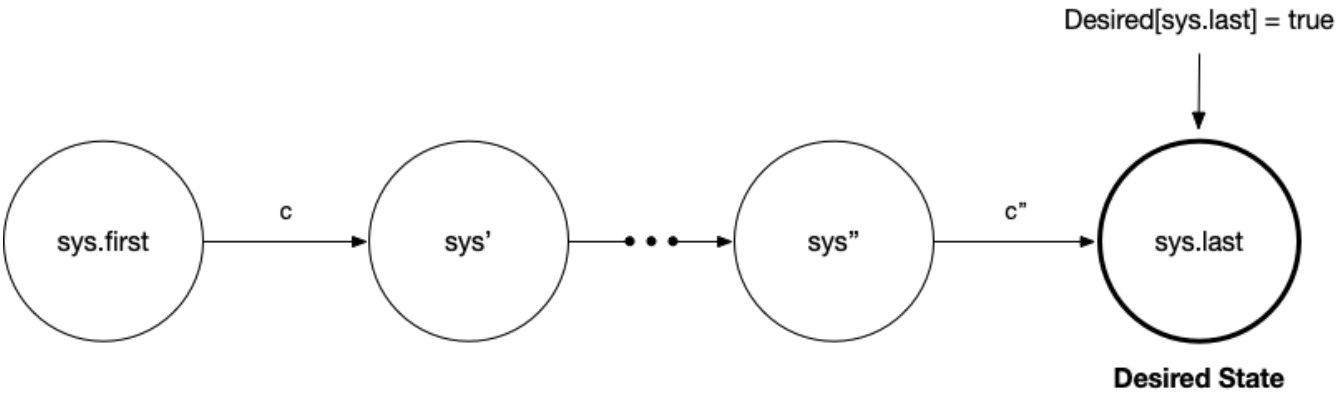


Figure 1. Cascading Commands

Figure 2. illustrates the resulting Command cascade after a Deployment Object has been submitted to the API Server by the user.

Is Kubernetes a declarative system?




```
fact {
    all sys : Sys - last | let sys' = sys.next {
        some c : Command {
            command.source = sys and command.target = sys'
        }
    }
    Desired[sys.last]
}
```

k8s-mechanics-declarative.als hosted with ❤ by GitHub view raw

Listing 6. State Transition Mechanics of a Declarative System

Listing 6. specifies the state transition mechanics of a declarative system. Given a Desired State Predicate, the system will determine a sequence of commands that transitions the system from its current state k8s.first to its desired state k8s.last.

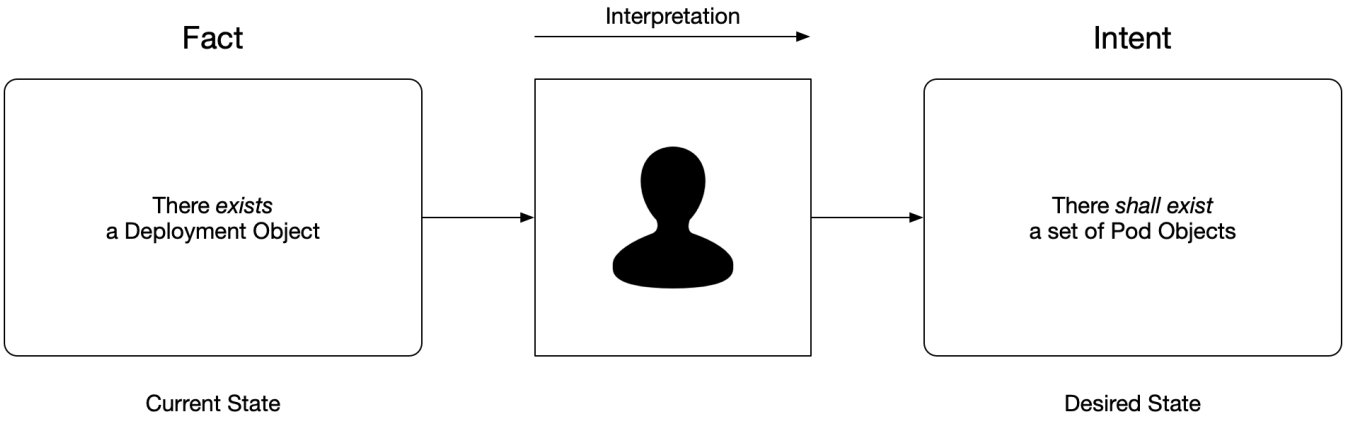


Figure 2. Interpreting Kubernetes Objects

We may subscribe to the notion that Kubernetes is a declarative system, if we interpret a Kubernetes Object not as a record of fact but as a record of intent:

For example we may interpret the Deployment Object from Listing 4. as the *intent* that there *shall exist* a set of 3 Pod Objects, so that the objects'

- `.spec.containers[0].image` equals `BusyBox`

However, this notion is not without subtleties: if you interpret an Object as a record-of-intent, you are presented with multiple options. For example, a Deployment Object may be interpreted as:

- there shall be a ReplicaSet or
- there shall be a set of Pods

Depending on the interpretation, the current state may or may not match the desired state

- if there is a ReplicaSet and *optionally* a set of Pods or
- if there is a ReplicaSet and *mandatorily* a set of Pods

Independent of our interpretation

- K8s is in steady state in relation to the Deployment Object if there is a ReplicaSet Object (the Deployment Controller will not yield Commands)
- K8s is in steady state in relation to the ReplicaSet Object if there is a set of Pod Objects (the ReplicaSet Controller will not yield Commands)

Conclusion

In casual conversations, Kubernetes may be described as a declarative system and Kubernetes Objects may be described as records-of-intent.

However, when you reason about Kubernetes and Kubernetes' behavior, you should keep in mind that Kubernetes does not make a coordinated effort to transition to a desired state. Instead, Kubernetes makes continuous, uncoordinated efforts to transition to a steady state.

About this post

This blog post is part of a collaborative effort between the CNCF, Google, and SAP to advance the understanding of Kubernetes and its underlying concepts.

Thanks to Andrew Chen.

Kubernetes