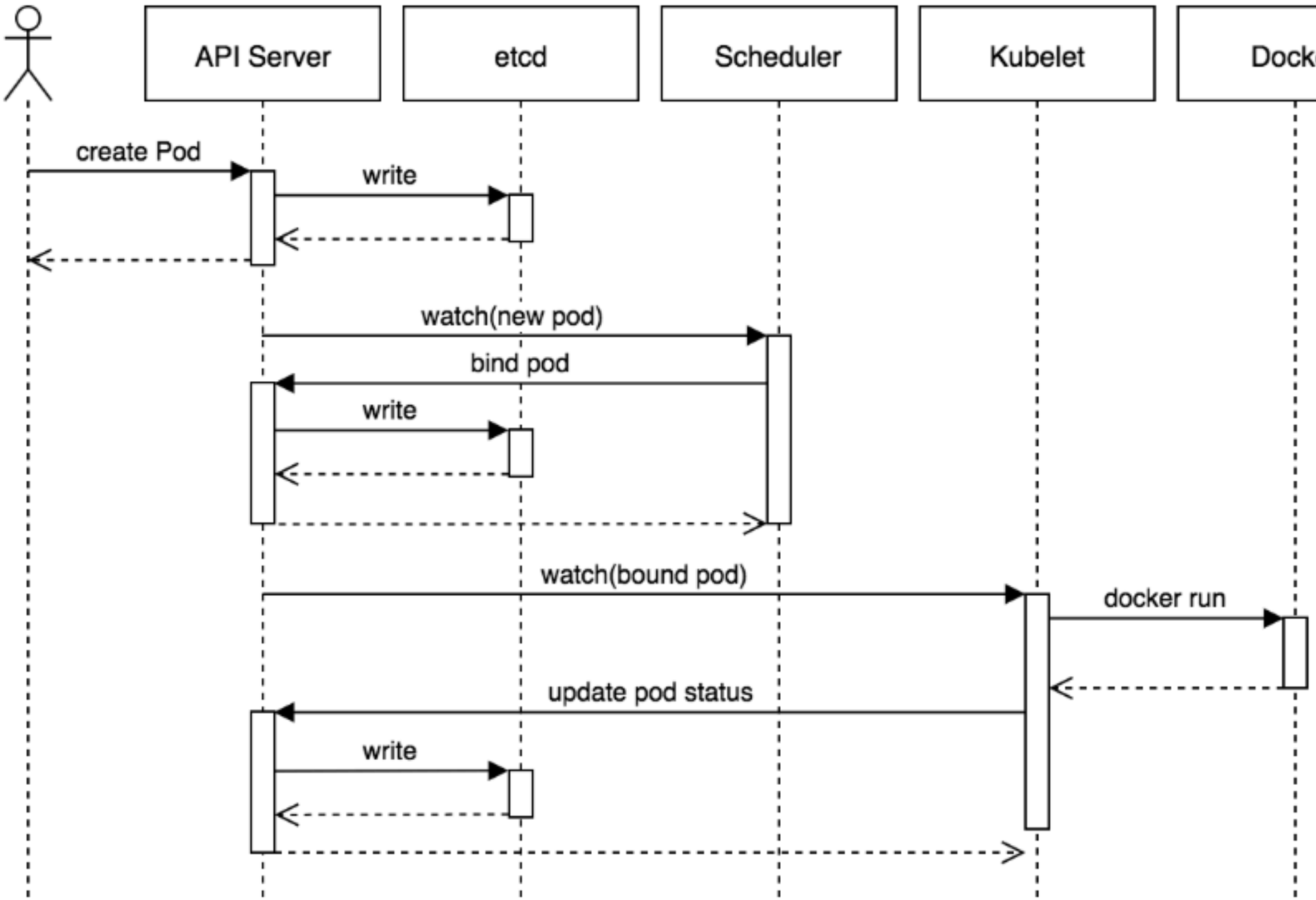
# kubernetes

*Kubernetes: Pod Lifecycle*

- A **Pod** is the **smallest unit of work** which can be scheduled in Kubernetes.
- A **Pod** encapsulates an application container(s), storage resources, unique network IP and options that govern how a container should run.
- Pods, Applications are generally deployed via higher level constructs such as Deployments, Replica Sets.
- Interaction with Pods is generally used to troubleshoot issues, hence understanding of Pods is important.

### States of a Pod

- **Pending:** The pod is accepted by the Kubernetes system but its container(s) is/are not created yet.
- **Running:** The pod is scheduled on a node and all its containers are created and at-least one container is in Running state.
- **Succeeded:** All container(s) in the Pod have exited with status 0 and will not be restarted.
- **Failed:** All container(s) of the Pod have exited and at least one container has returned a non-zero status.
- **CrashLoopBackoff:** The container fails to start and is tried again and again.


API Server
etcd
Scheduler
Kubelet
Docker
create Pod
write
watch(new pod)
bind pod
write
watch(bound pod)
docker run
update pod status
write

## Activities During a Pod's Life

- **Init Container:** Init containers are containers which are run before the main application container gets started.
- They have **two important characteristics**:
  They always run to completion.
  Each init container must complete before the next one is started.
- **Init containers** support all the fields and features of app containers, including resource limits, volumes, and security settings.
- **Init containers** can be useful when some initial actions need to be run before the main container in the pod starts.

## Lifecycle Hooks

- **LifeCycle Hooks** allows the user to run specific code during specific events of a containers lifecycle.
- **PostStart**: This hook gets executed upon container creation but there is no guarantee that it will run after the container ENTRYPOINT.
- **PreStop**: This hook gets executed just before a container is terminated. This is a blocking call which means the hook execution must complete before the call to delete a container can be sent.
- There are two types of **handlers** which can be implemented in the hook implementation:
  **Exec:** runs a specific command inside the container and the resources consumed by the command are counted against the container.
  **HTTP:** executes an HTTP request against a specific endpoint on the container.

## *Will see you in Next Lecture…*


*See you in next lecture …*