mahisoft

Kubernetes Cluster Management in Google Cloud

What is this about?

- Advanced Kubernetes Topics
- Internal structure of Kubernetes components
- Deployment of Clusters
- Day-to-day operation
- Cluster Scaling
- Application Scaling
- High Availability
- Ingress Controllers
- TLS Automation with ACME
- Much more

What is this about?

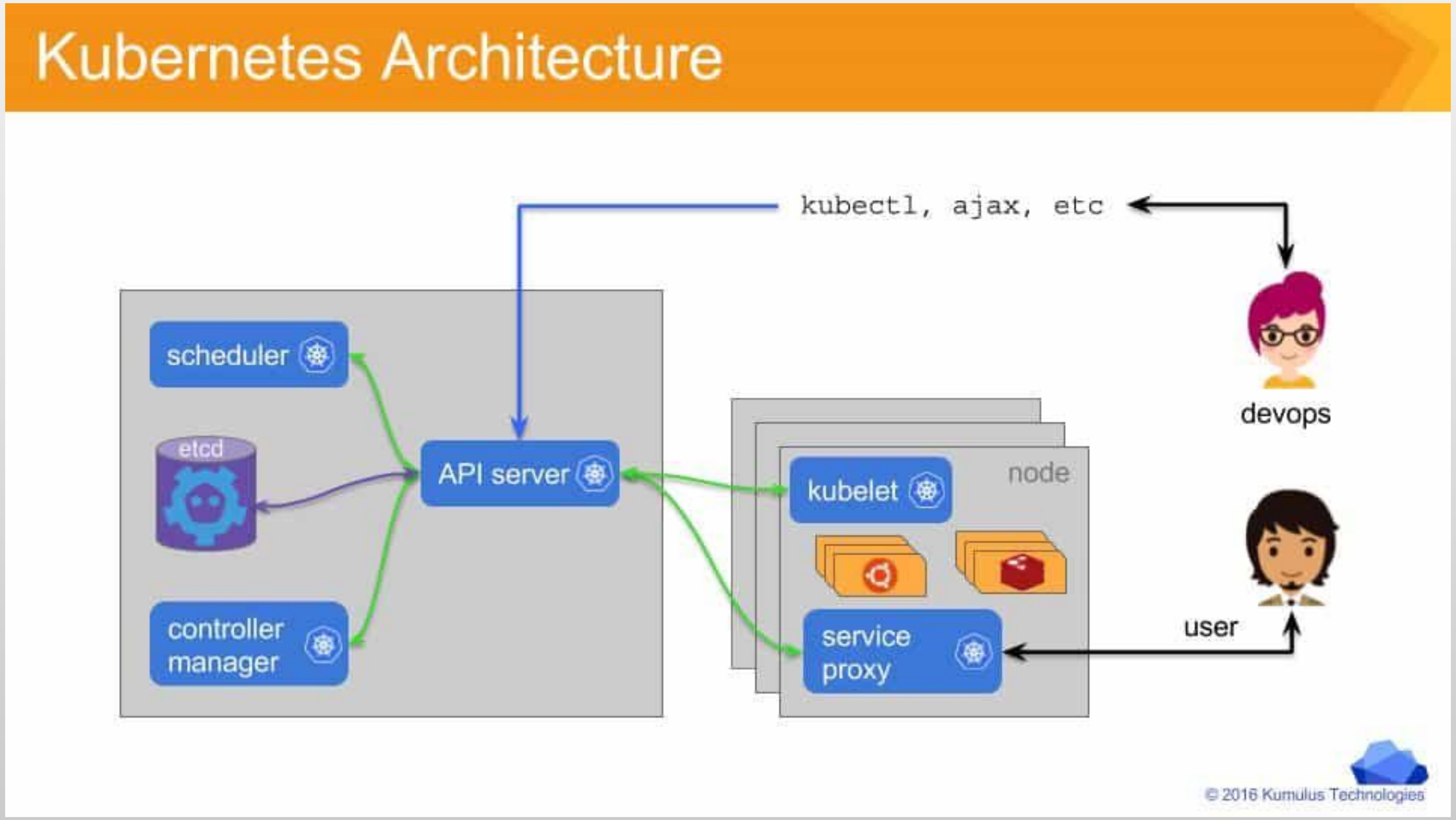
- Assignments will be given every week, they must be delivered by **monday** of the next week.
- There will be a quiz one week after the final mentoring session.

What is this about?

- Assignments will be given every week, they must be delivered by **monday** of the next week.
- There will be a quiz one week after the final mentoring session.

Session 1:
Kubernetes Internals

The components



etcd

- etcd is a distributed key-value store designed to reliably and quickly preserve and provide access to critical data.
- It enables reliable distributed coordination through distributed locking, leader elections, and write barriers.
- An etcd cluster is intended for high availability and permanent data storage and retrieval.
- Is used as Kubernetes' backing store for all cluster data.
- This means that everything that represents the cluster's state, is stored in etcd.

api-server

- The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others.
- The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

scheduler

- The Kubernetes scheduler is a policy-rich, topology-aware, workload-specific function that significantly impacts availability, performance, and capacity.
- The scheduler needs to take into account individual and collective resource requirements, quality of service requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, deadlines, and so on.

controller-manager

- The Kubernetes controller manager is a daemon that embeds the core control loops shipped with Kubernetes.
- A control loop is a non-terminating loop that regulates the state of the system.
- In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.
- Examples of controllers that ship with Kubernetes today are the replication controller, endpoints controller, namespace controller, and serviceaccounts controller.

kubelet

- The kubelet is the primary “node agent” that runs on each node.
- The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod.
- The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy.
- The kubelet doesn’t manage containers which were not created by Kubernetes.

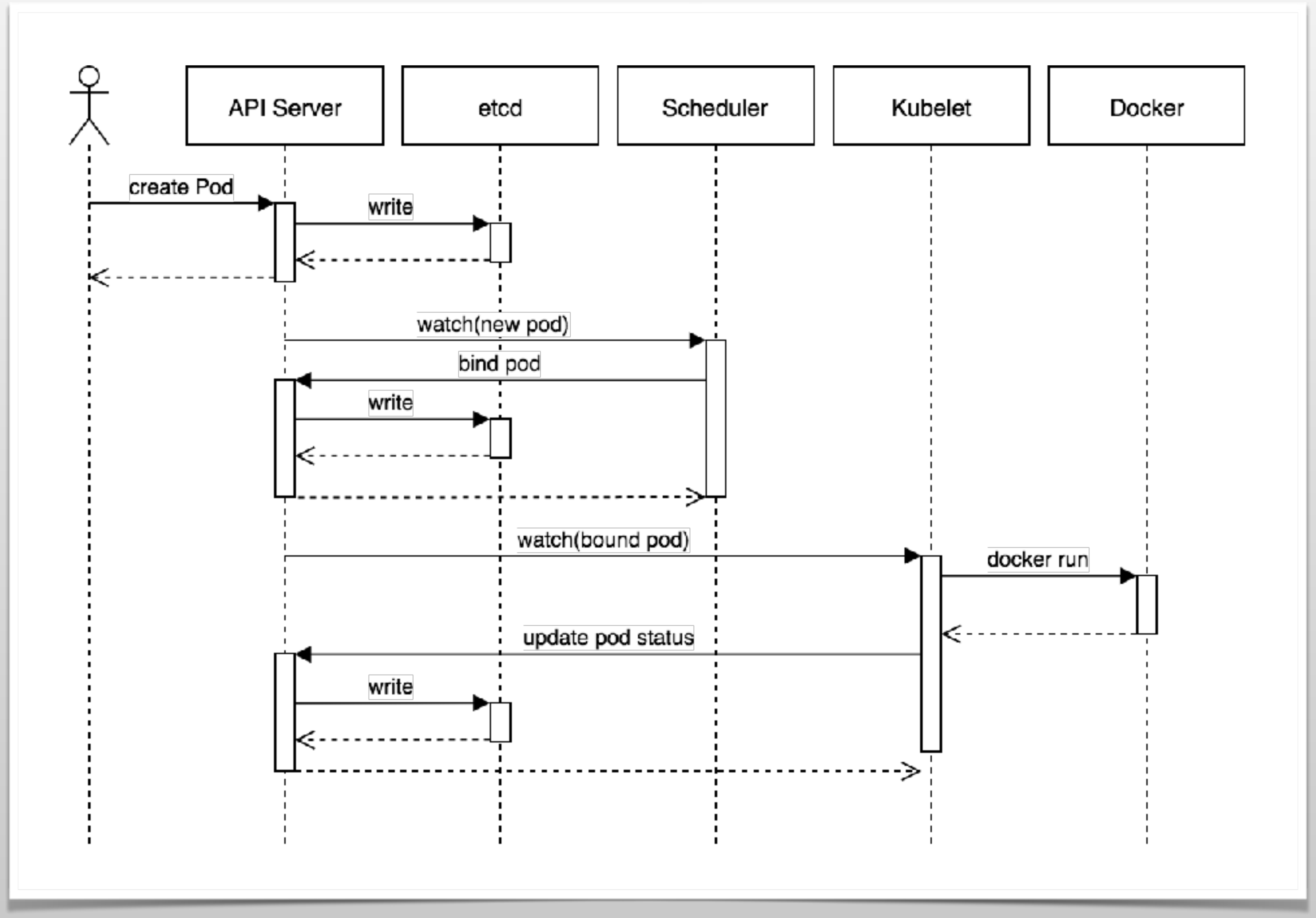
kube-proxy

- The Kubernetes network proxy runs on each node. This reflects services as defined in the Kubernetes API on each node and can do simple TCP and UDP stream forwarding or round robin TCP and UDP forwarding across a set of backends.
- Service cluster IPs and ports are currently found through Docker-links-compatible environment variables specifying ports opened by the service proxy.
- There is an optional addon that provides cluster DNS for these cluster IPs. The user must create a service with the apiserver API to configure the proxy.

Communication

- All communication between components (even between etcd cluster members) is encrypted and secured with TLS.
- Trust is stablished with certificates, with all components allowing the configuration of a CA certificate that will be trusted.
- The usual configuration is to create a single CA and emit certificates to all the desired components with it. Then each component will be configured to trust said CA.

Communication



Architecture

- These components are deployed in two type of nodes:
  - Master nodes run etcd, api-server, scheduler and controller-manager.
    - There can be 1…N master nodes
    - All the nodes run all the services for HA
  - Worker nodes run kubelet and kube-proxy
    - There can be 0…M worker nodes
    - Each node runs both kubelet and kube-proxy, but in each node those serve the containers running within the same node.

Architecture

- Typically, worker nodes and master nodes do not overlap.
- However, if you’re resource constrained you could deploy a single node that acts like both. However:
  - A failure in the workload (aka the pods) will cause the k8s api to become unresponsive, or worse, to act accordingly to drive the cluster towards a healthy state
  - A falure in k8s might hinder your workload, making it unresponsive (which is no good)
- Having them in separate nodes separates concerns and failures on one domain will not affect the other one.

Architecture

- If you’re rolling your own cluster, you must have at least **three** master nodes.
  - This is the minimum that etcd states for proper functioning.
- The amount of worker nodes will depend on your workload’s necessities.
  - If you have a good idea of resource request/limits you will require, this table is a good place to start.

Architecture

- Most cloud providers that offer k8s as a provisioned service will assume the cost of the master(s).
- Use this to your advantage to reduce operating costs.

mahisoft

Questions?

Assignment

- Create your own Kubernetes cluster, from scratch.
- Follow this guide:
  - https://github.com/kelseyhightower/kubernetes-the-hard-way
- You can get $300 worth of credit on Google Cloud when you create an account (this should be enough for the assignment).
- When you finish, send me the output of these commands:
  - kubectl get componentstatuses
  - kubectl config view
  - kubectl describe nodes