KubeCon | CloudNativeCon

North America 2018

Kubernetes Design Principles: Understand the Why

December 12, 2018
Saad Ali
Senior Software Engineer, Google
Co-Lead SIG-Storage
github.com/saad-ali
twitter.com/the_saad_ali

What’s in it for me?

- Goal: A deeper understanding of Kubernetes
- Important tool for learning
    - Understand the problem
    - The “why” not just the “what”

Kubernetes

Containerization was the key

- Consistent, repeatable, reliable deployments on a wide variety of systems.

Who will manage it?

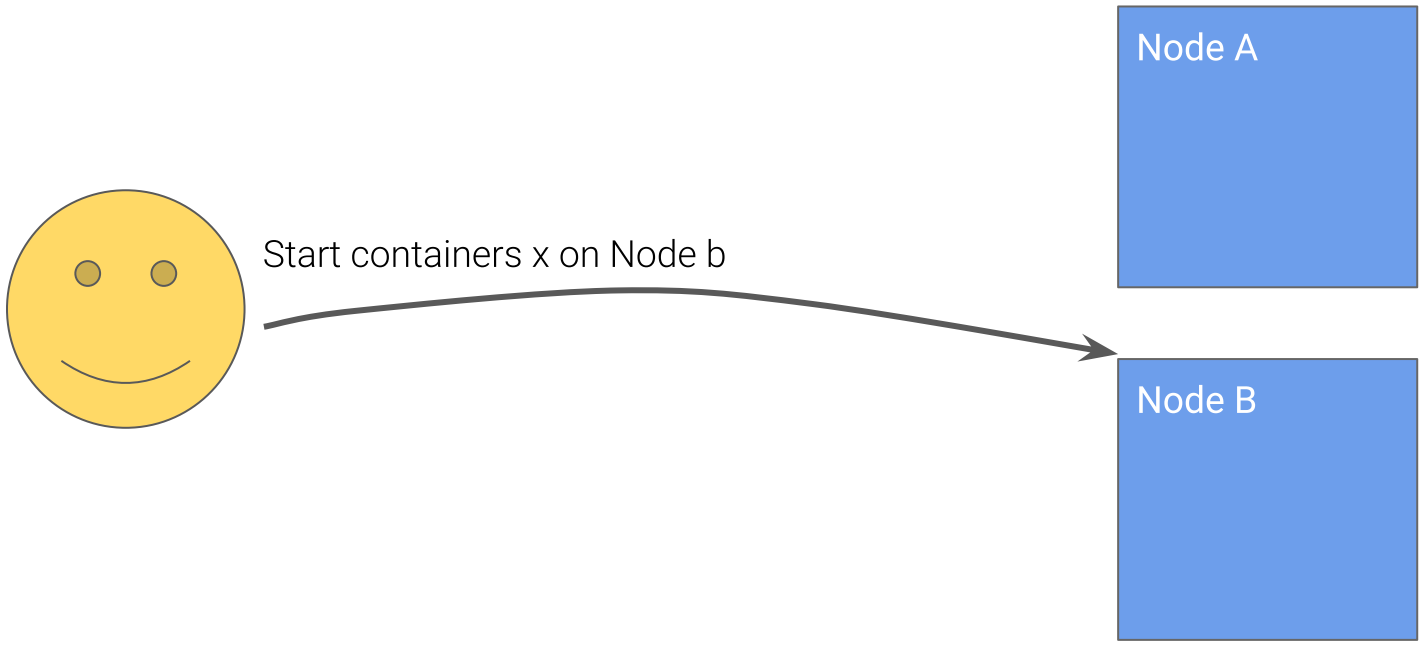
- You? Scripts? A system you write?

Kubernetes manages your cluster!

- Deploys & monitors containerized workloads.

How to deploy a workload?

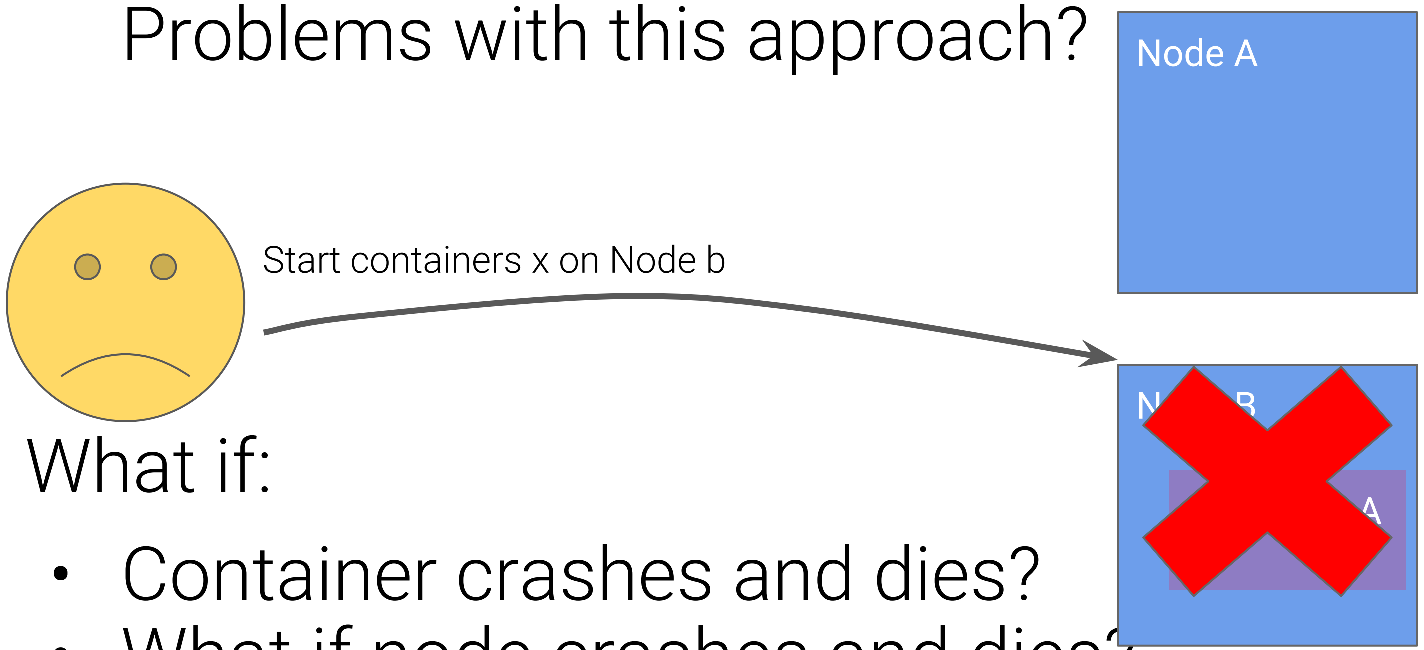
Obvious solution



How to deploy a workload?

Obvious solution

Problems with this approach?



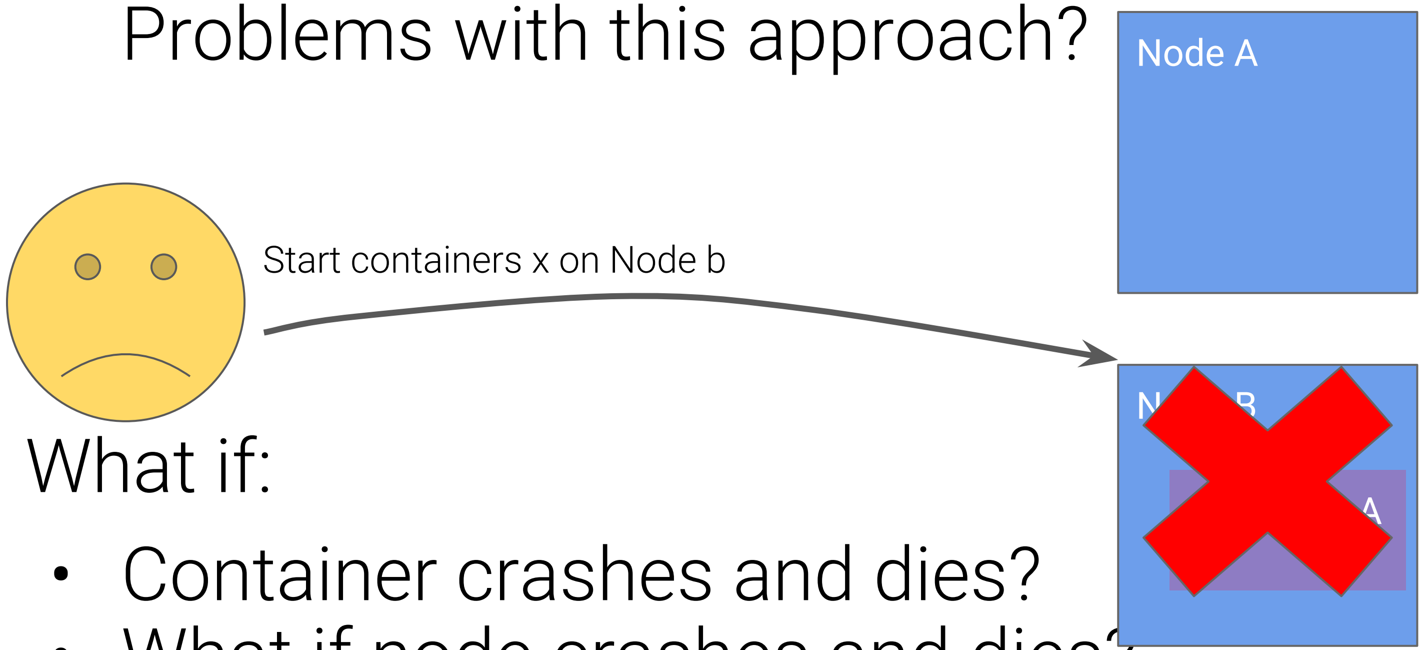
What if:

- Container crashes and dies?
- What if node crashes and dies?
- What if node B had a momentary blip?

How to deploy a workload?

Obvious solution

Problems with this approach?



What if:

- Container crashes and dies?
- What if node crashes and dies?
- What if node B had a momentary blip?

User has to

- Monitor and store state of every container/node.
- "Catch up" any failed nodes that missed calls.

Complex, custom logic.

Principle #1

Kubernetes APIs are *declarative* rather then *imperative*.

Declarative APIs

Before:

- **You:** provide exact set of instructions to drive to desired state
- **System:** executes instructions
- **You:** monitor system, and provide further instructions if it deviates.

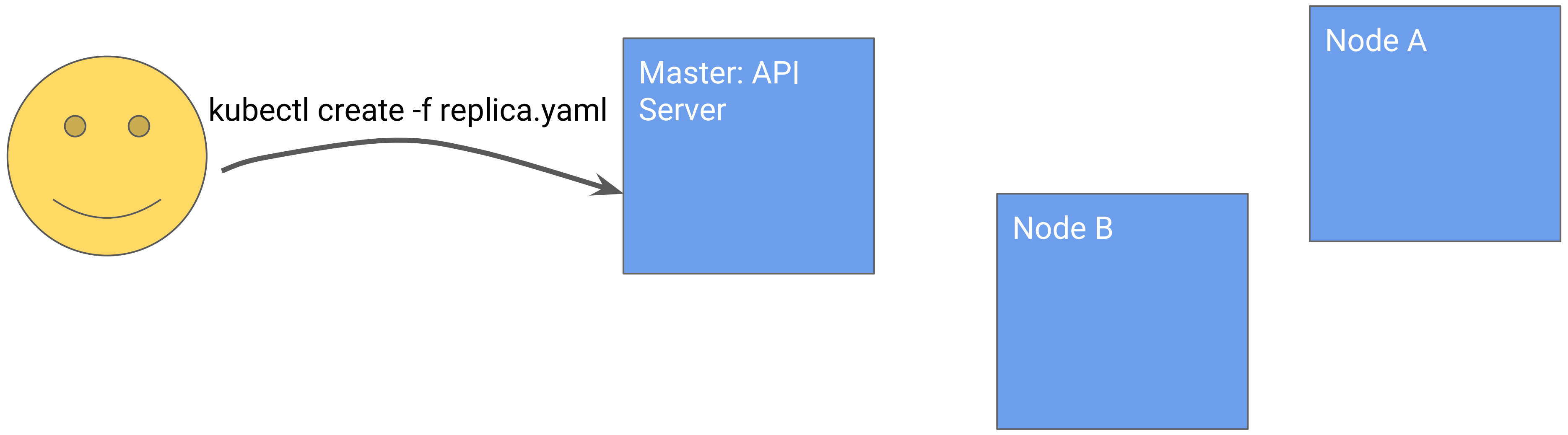
After:

- **You:** define desired state
- **System:** works to drive towards that state

How to deploy a workload?

The Kubernetes way!

- **You:** create API object that is persisted on kube API server until deletion
- **System:** all components work in parallel to drive to that state

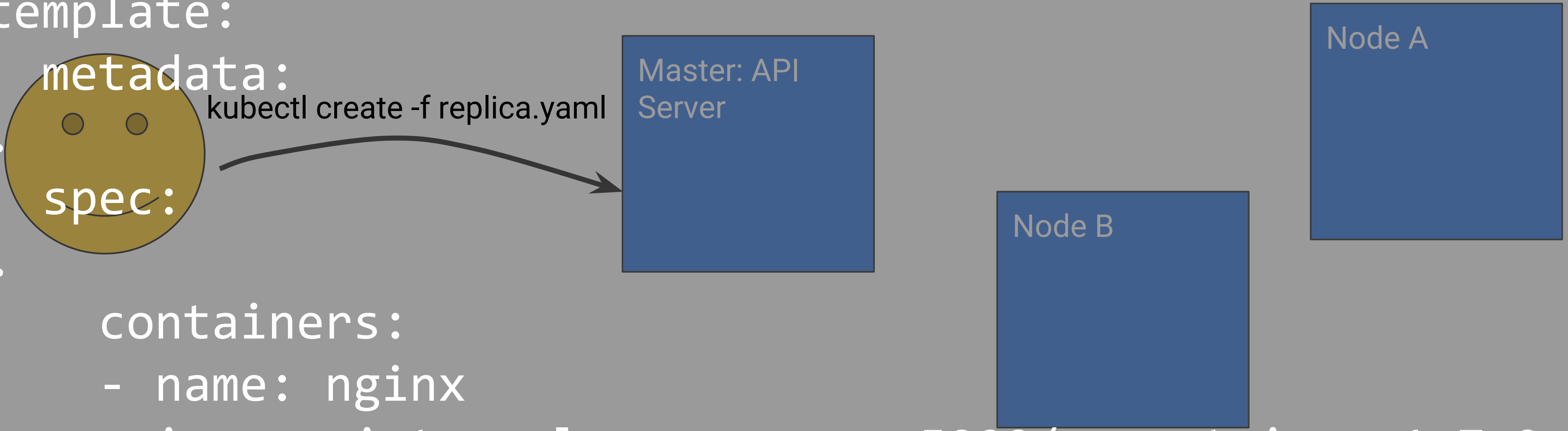


How to deploy a workload?

The Kubernetes way!

- **You:** create API object that is persisted on kube API server until deletion
- **System:** all components work in parallel to drive to that state

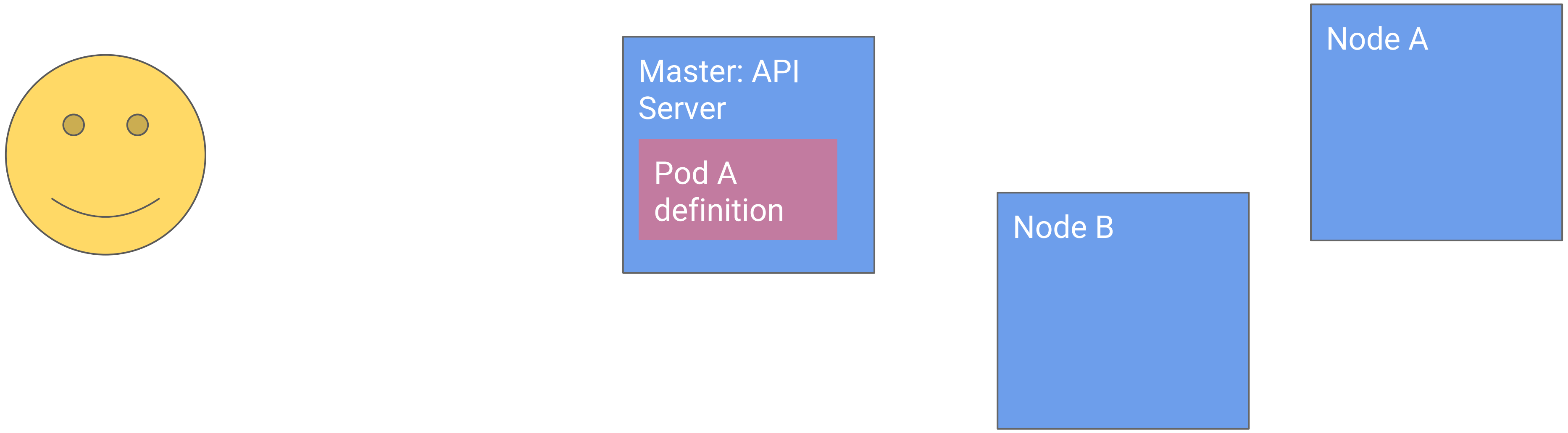
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 1
  template:
    metadata:
...
    spec:
...
      containers:
      - name: nginx
        image: internal.mycorp.com:5000/mycontainer:1.7.9
```



How to deploy a workload?

The Kubernetes way!

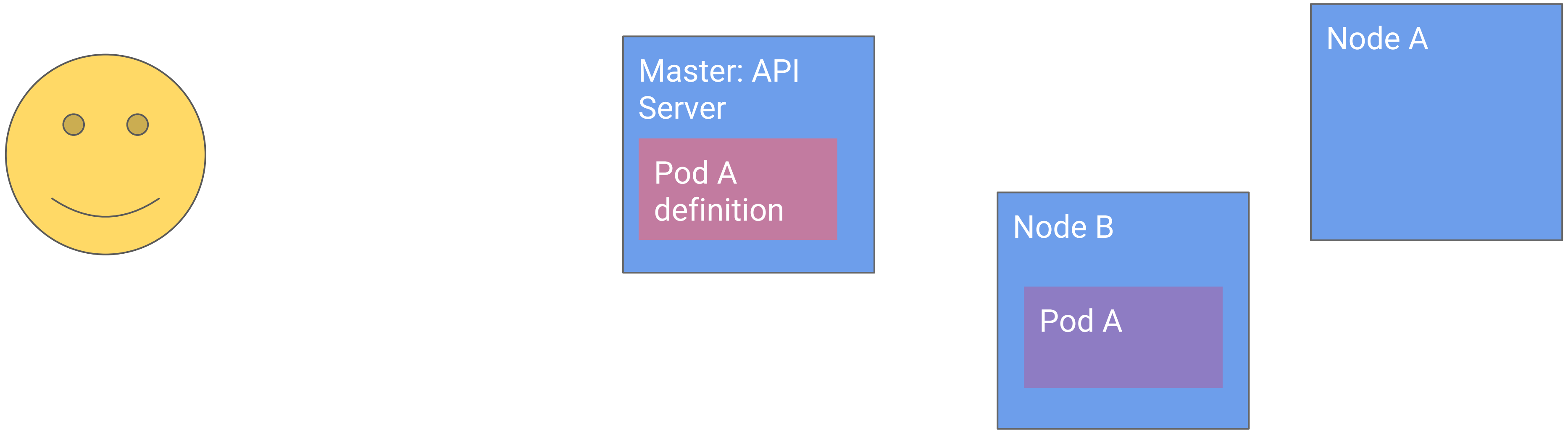
- **You:** create API object that is persisted on kube API server until deletion
- **System:** all components work in parallel to drive to that state



How to deploy a workload?

The Kubernetes way!

- **You:** create API object that is persisted on kube API server until deletion
- **System:** all components work in parallel to drive to that state



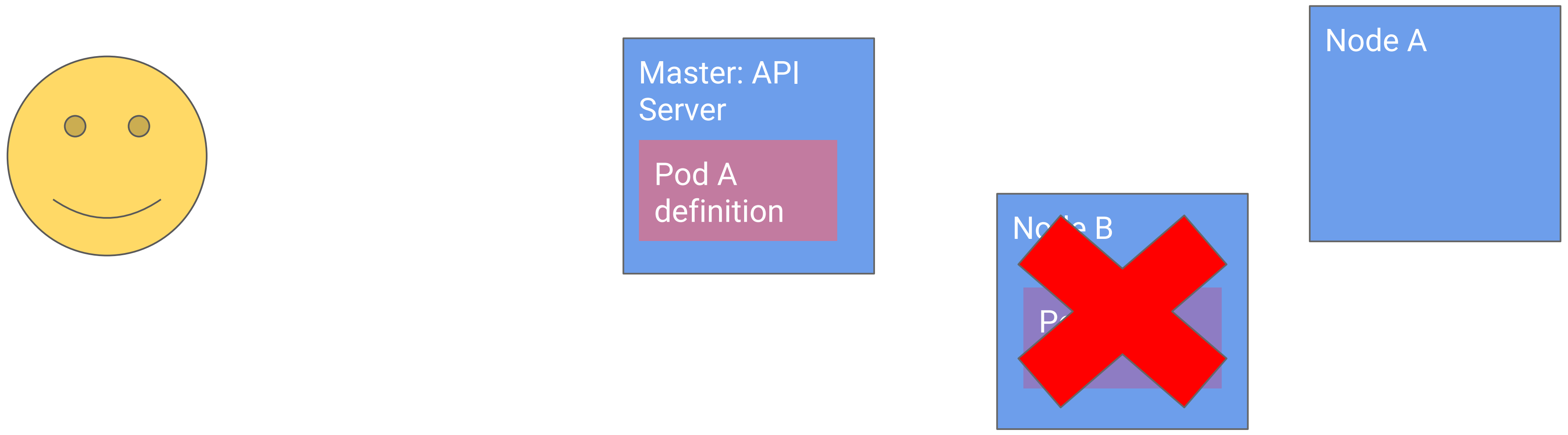
Why declarative over imperative?

Automatic recovery!

Why declarative over imperative?

Example:

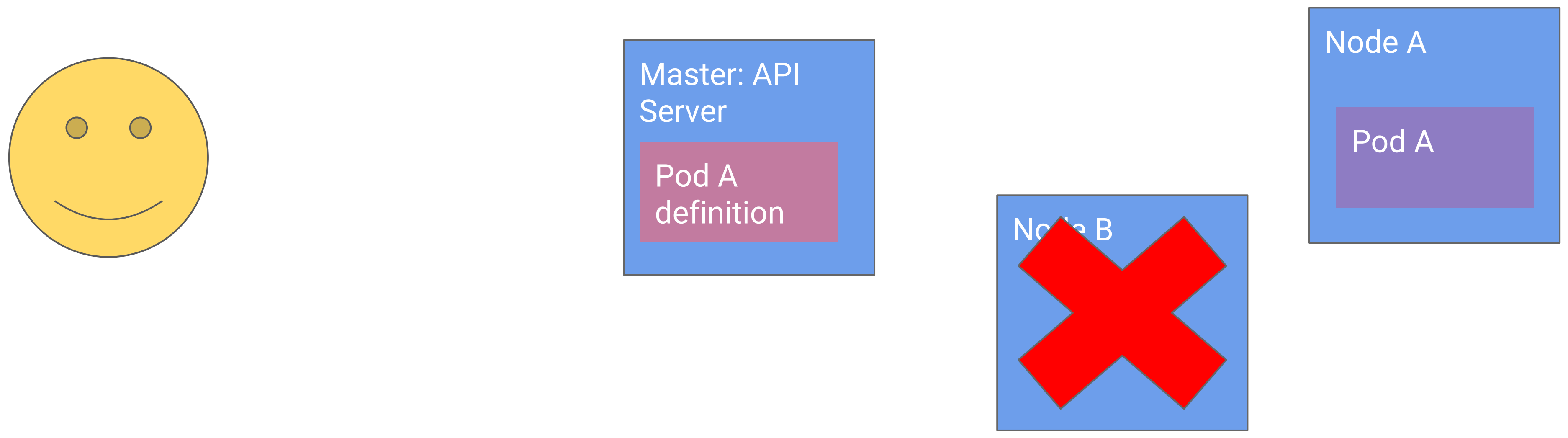
- Step 1: Node failure.



Why declarative over imperative?

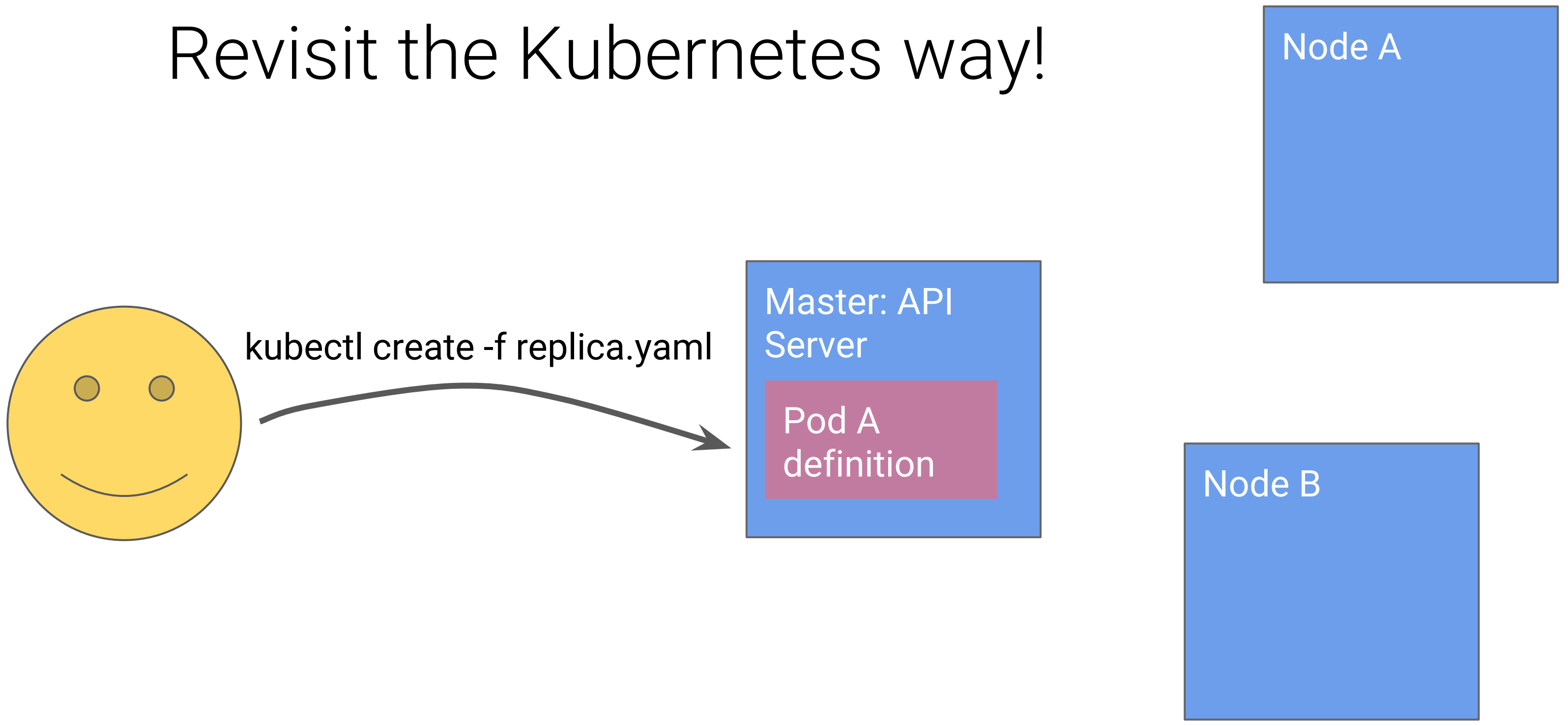
Example:

- Step 1: Node failure.
- Step 2: System automatically moves pod to healthy node.



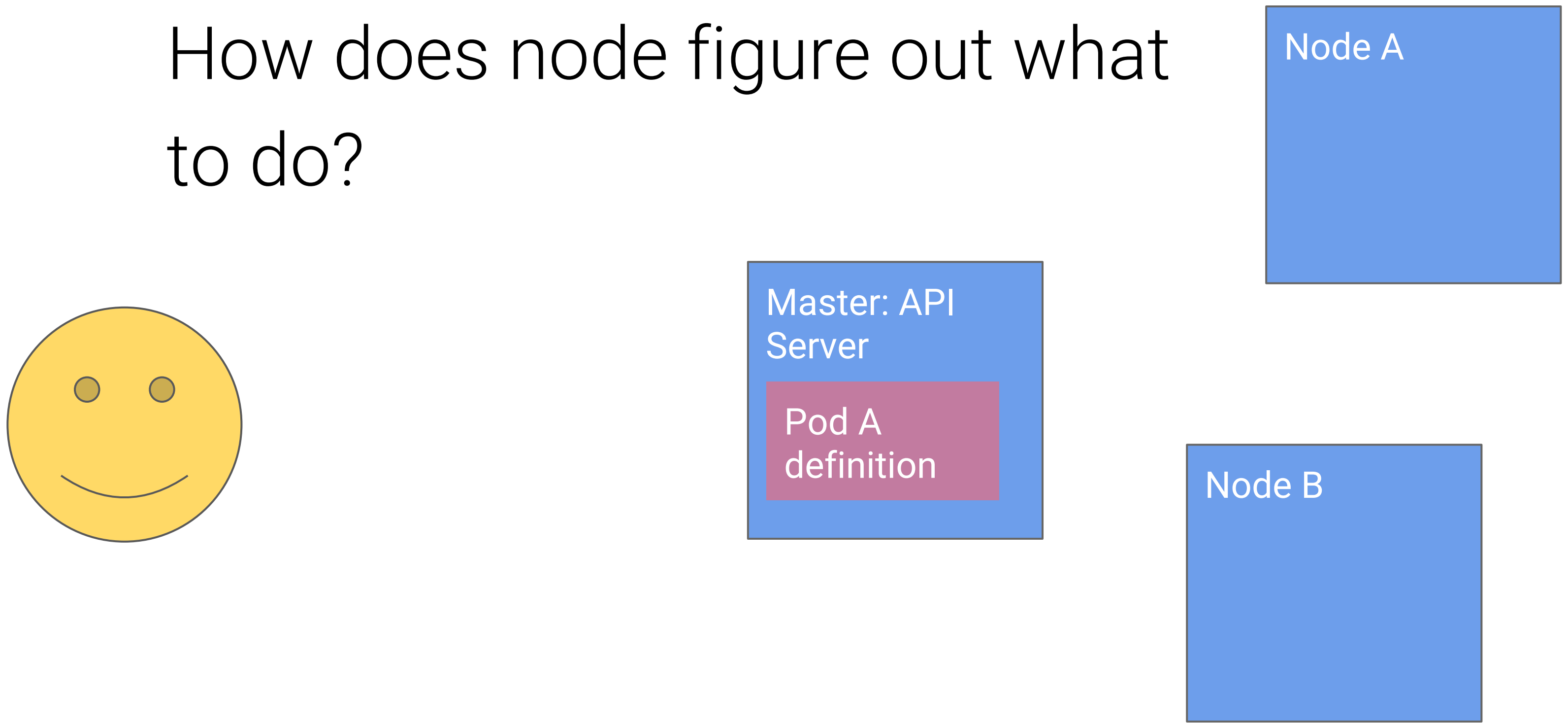
How to deploy a workload?

Revisit the Kubernetes way!



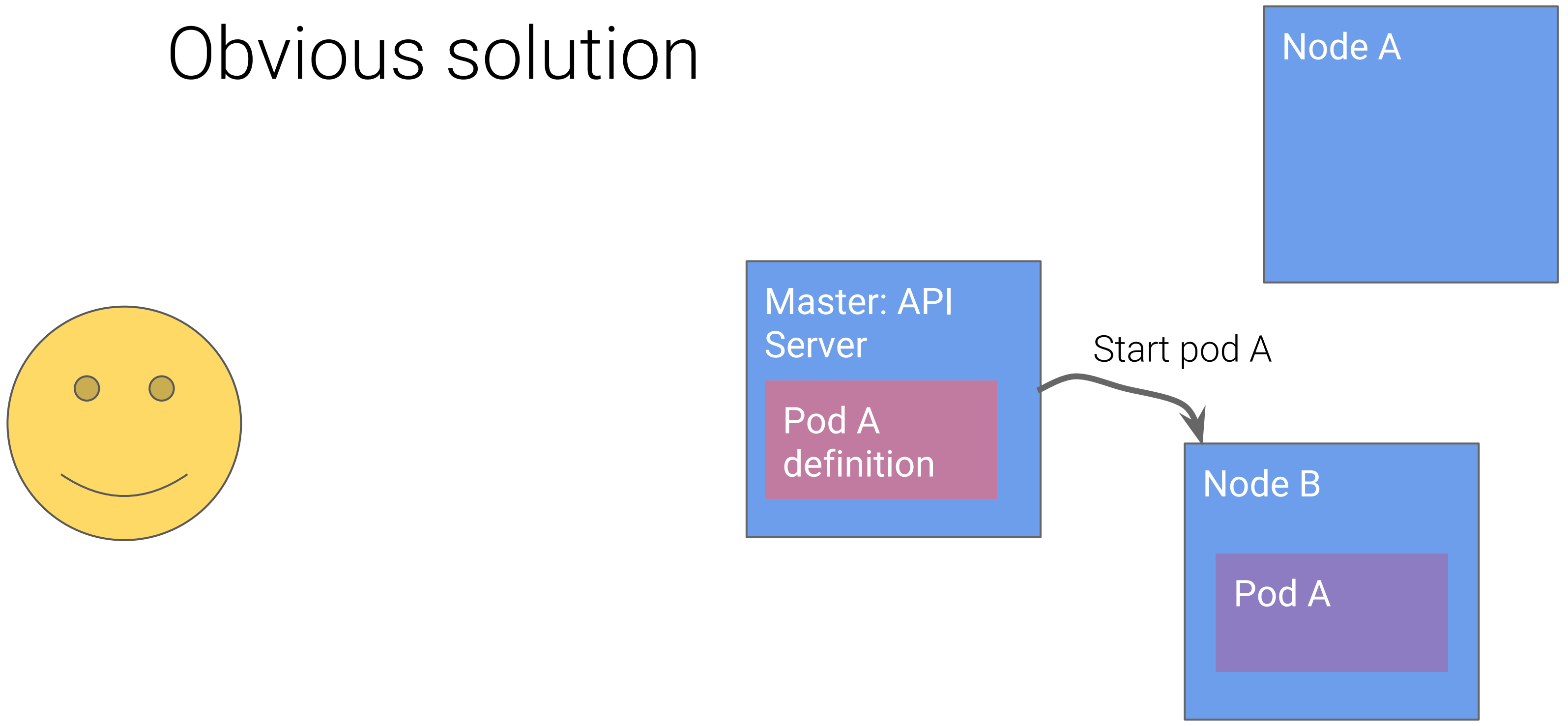
How to deploy a workload?

How does node figure out what to do?



How to deploy a workload?

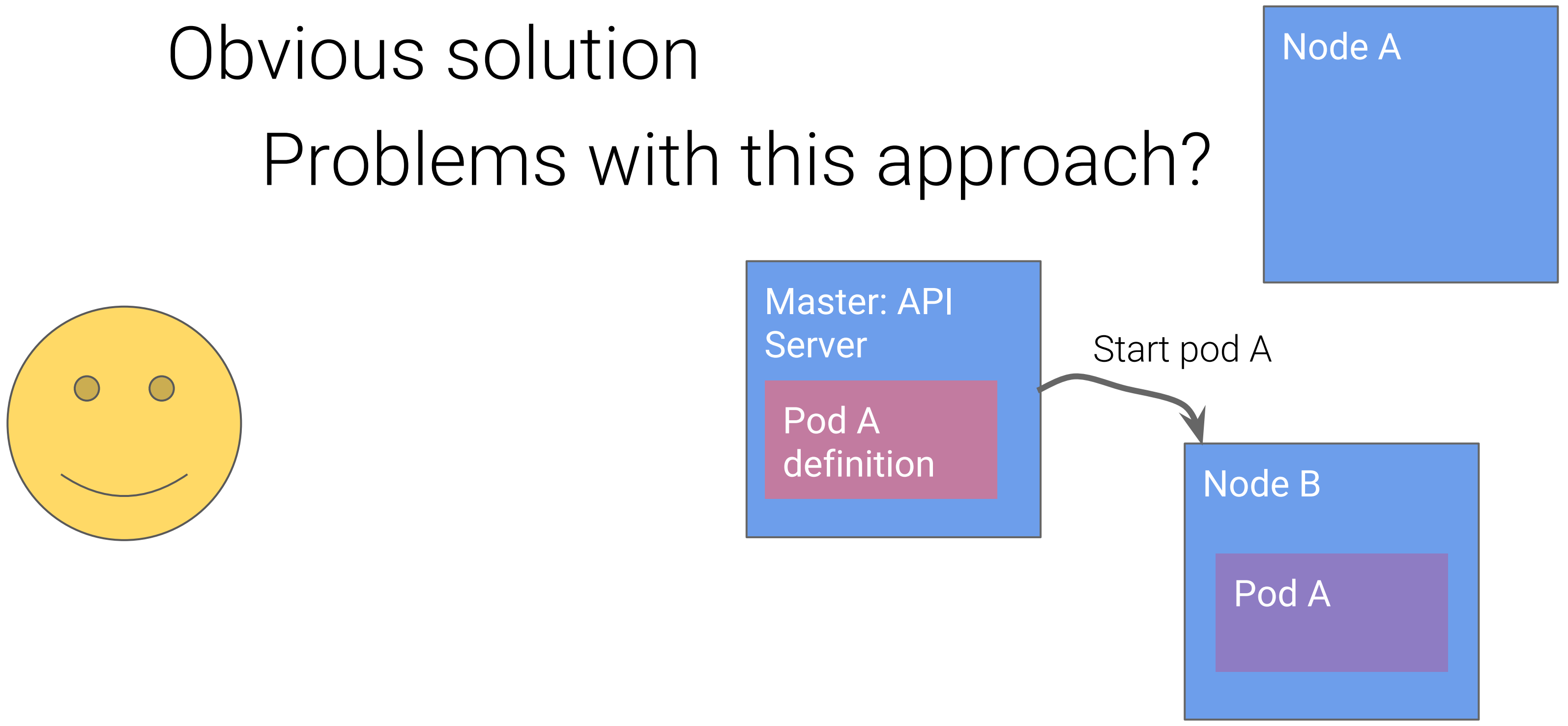
Obvious solution



How to deploy a workload?

Obvious solution

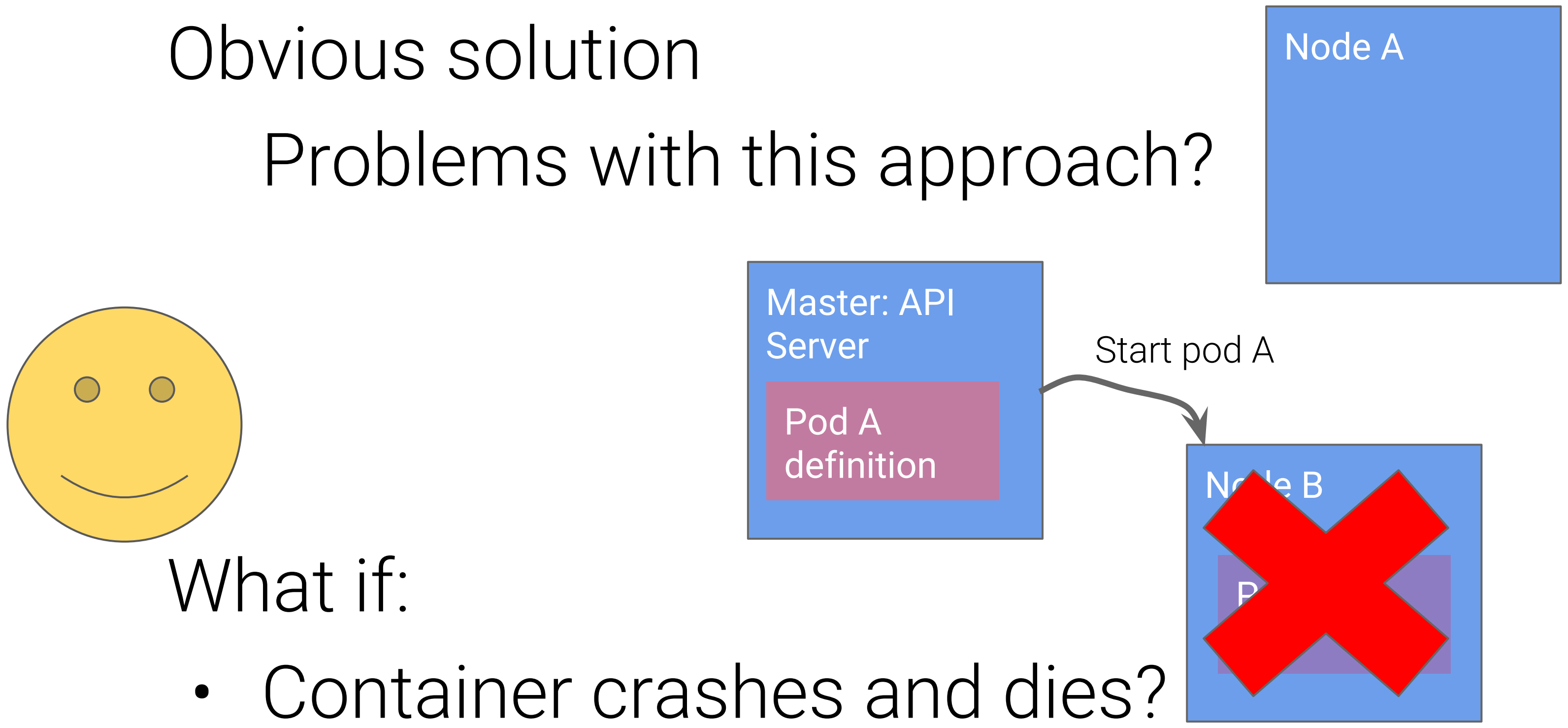
Problems with this approach?



How to deploy a workload?

Obvious solution

Problems with this approach?



What if:

- Container crashes and dies?
- What if node crashes and dies?
- What if node B had a momentary blip?

Master has to

- Monitor and store state of every component.
- "Catch up" any failed components that missed calls.

Master becomes:

- Complex
- Brittle
- Difficult to extend

Principle #2

The Kubernetes control plane is transparent.
There are no hidden internal APIs.

No hidden internal APIs

Before:

- **Master:** provides exact set of instructions to drive node to desired state
- **Node:** executes instructions
- **Master:** monitors nodes, and provides further instructions if state deviates.

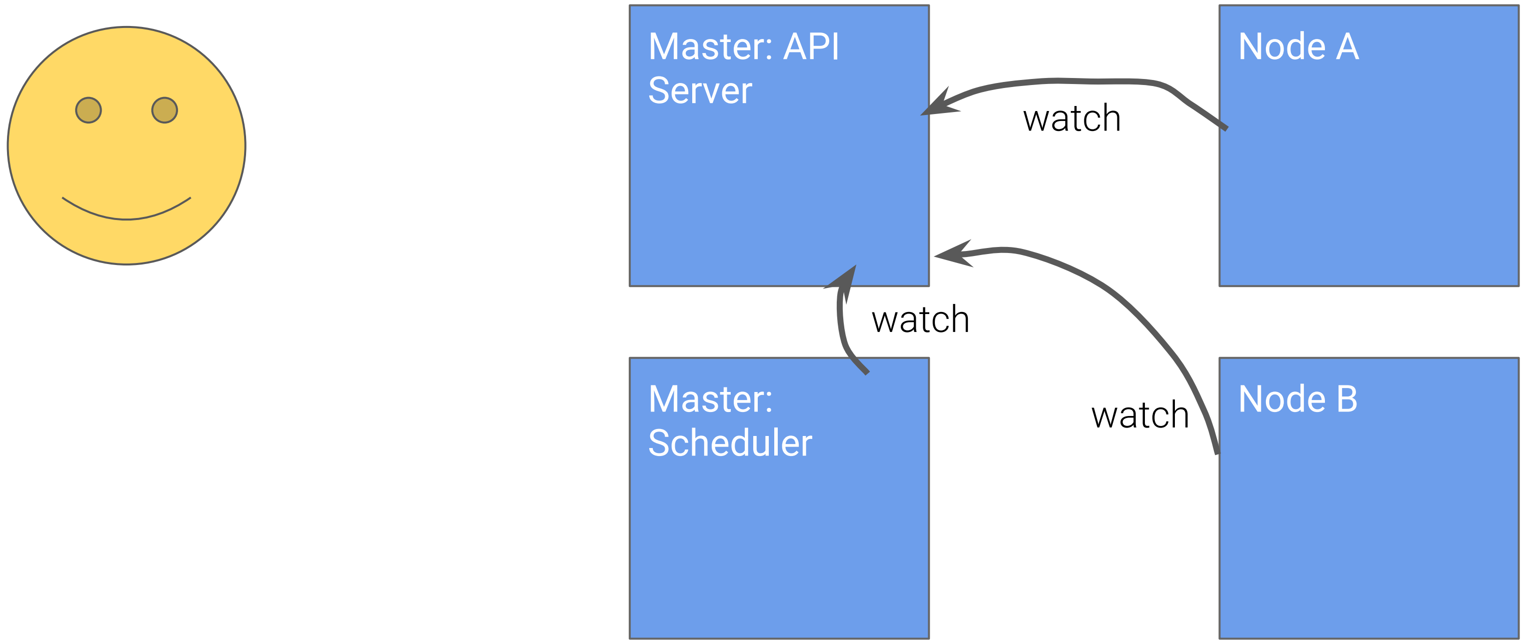
After:

- **Master:** defines desired state of node
- **Node:** works independently to drive itself towards that state

No hidden internal APIs

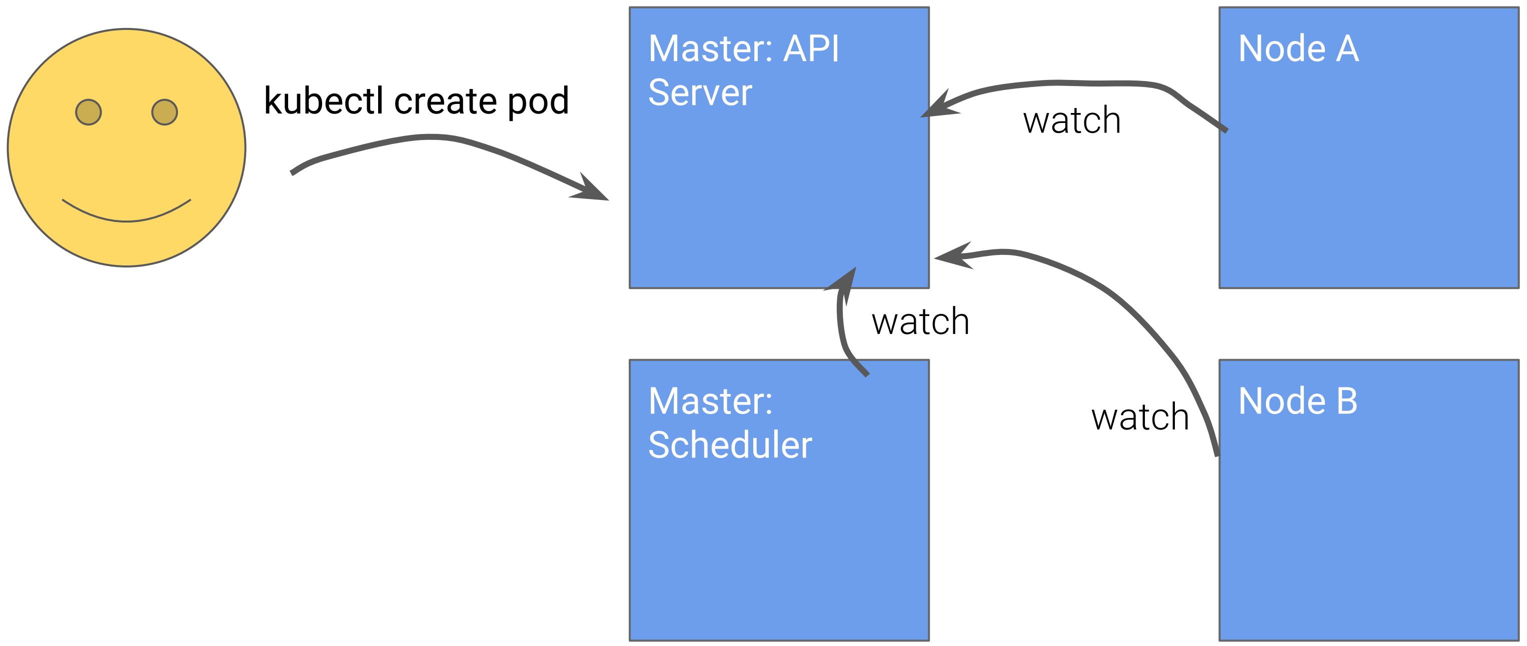
KubeCon | CloudNativeCon
North America 2018

All components watch the Kubernetes API, and figure out what they need to do.



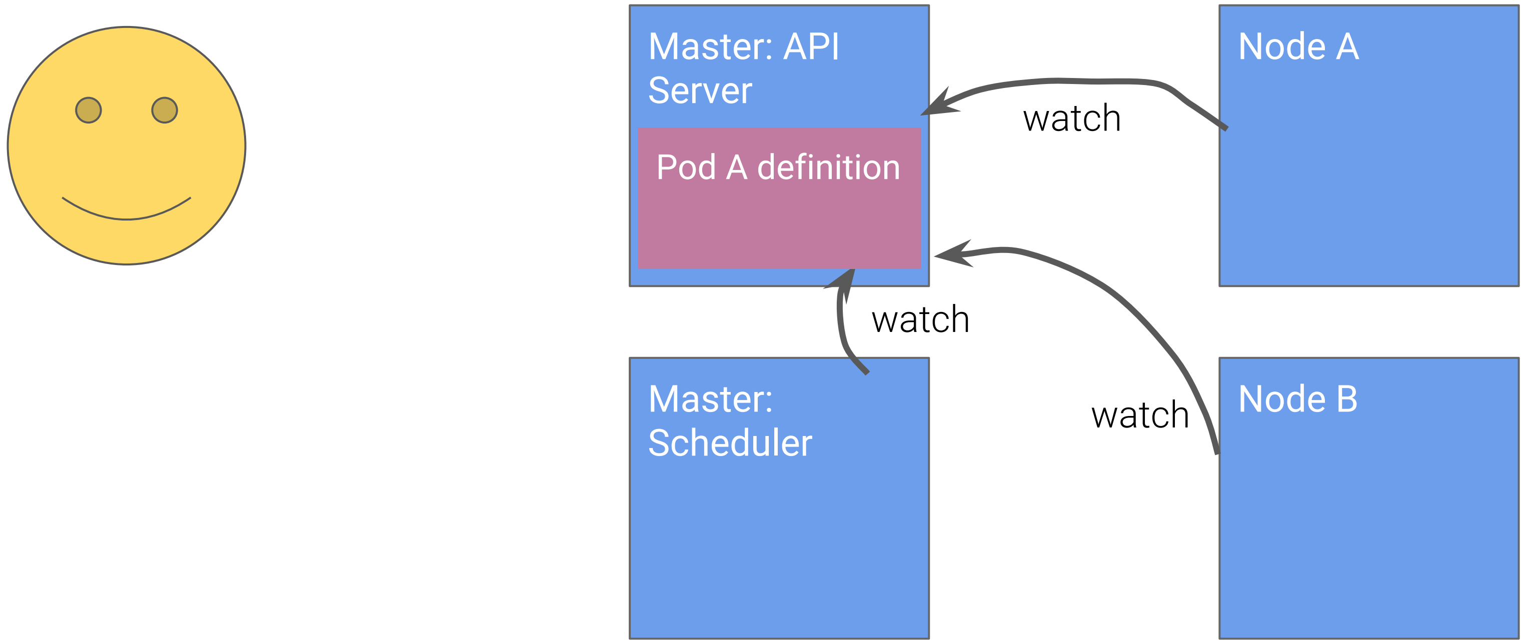
No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



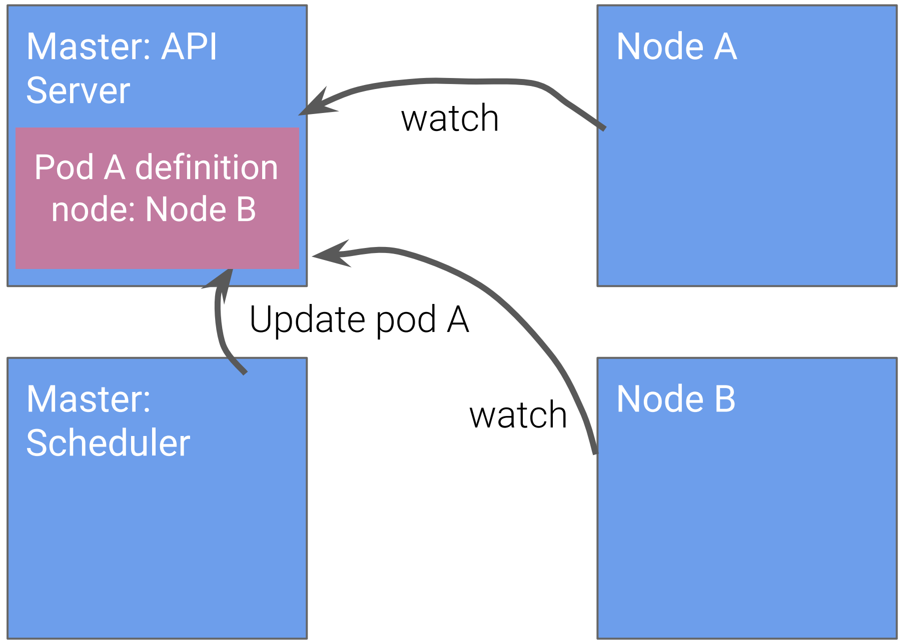
No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



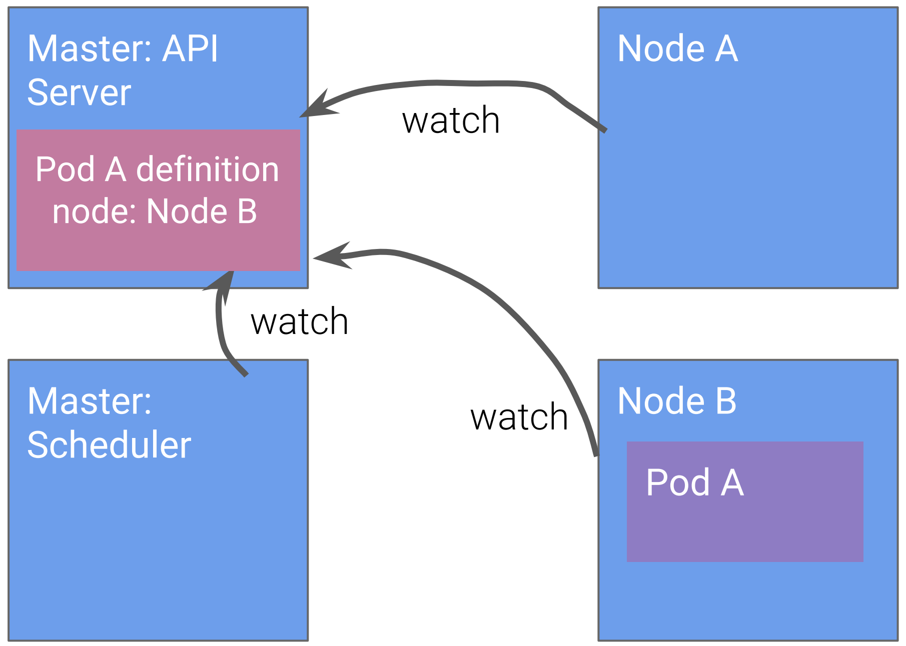
No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



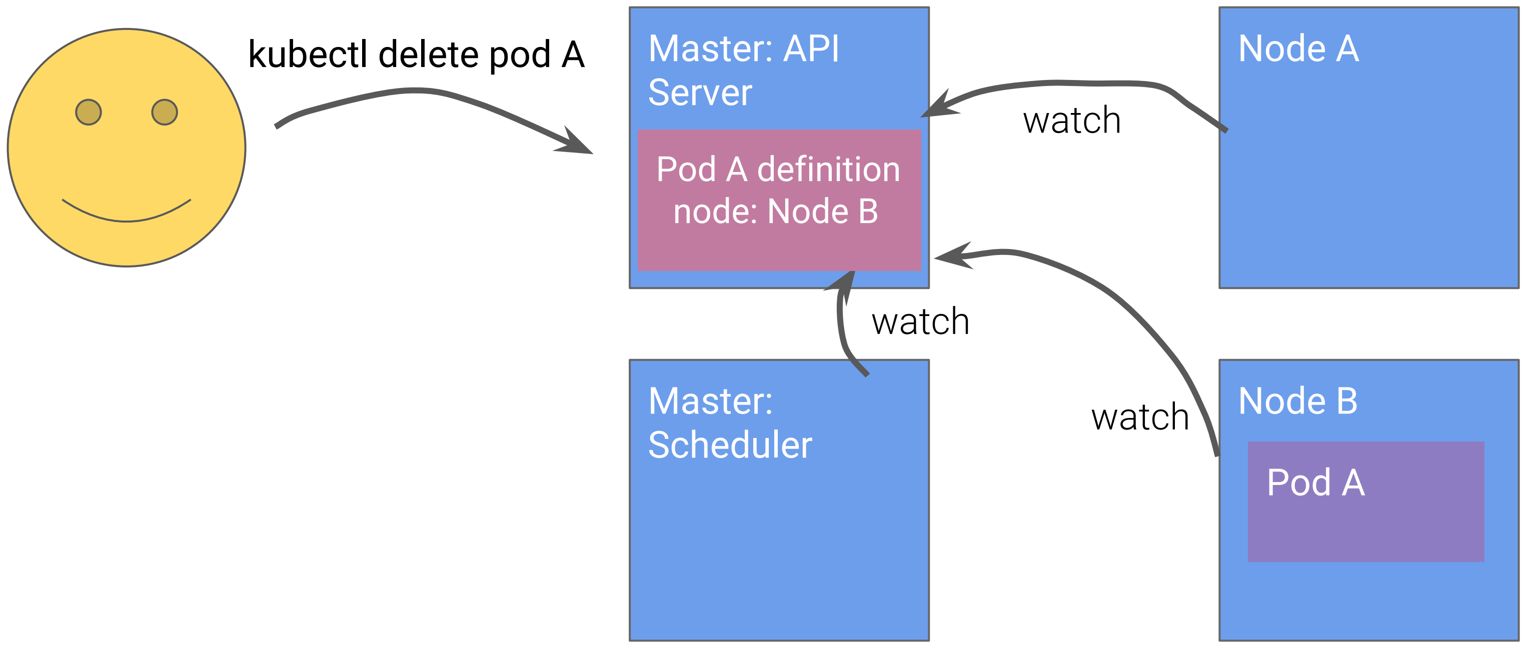
No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



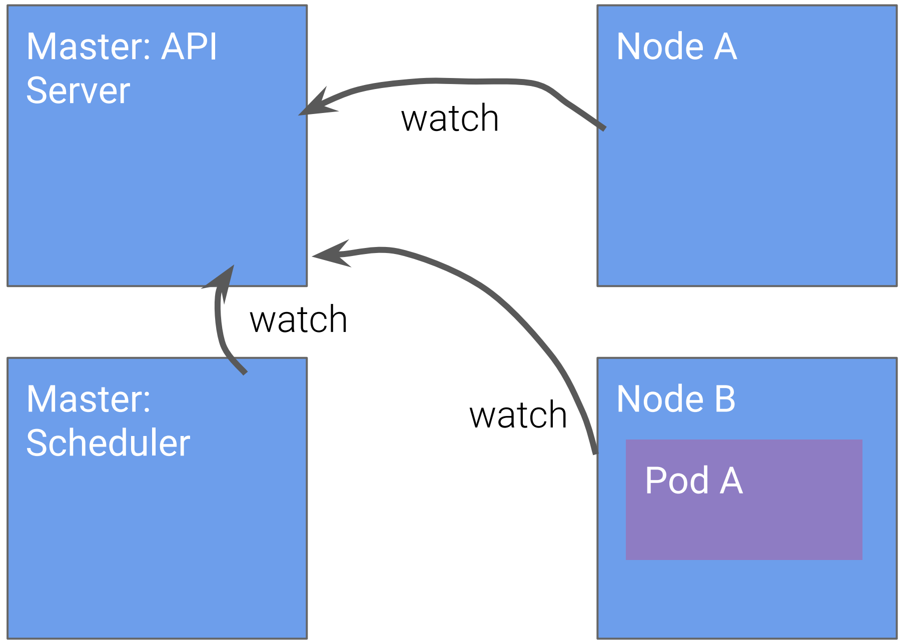
No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



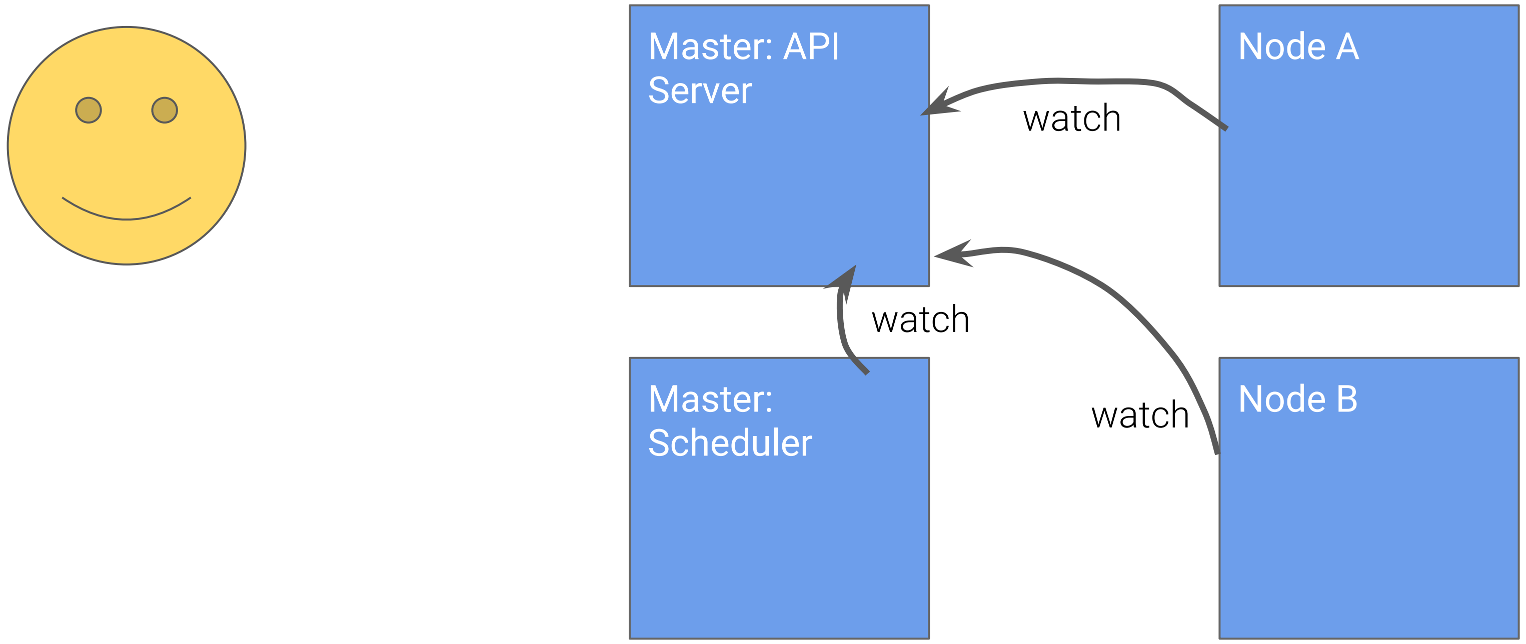
No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



No hidden internal APIs

All components watch the Kubernetes API, and figure out what they need to do.



Why No hidden internal APIs?

Declarative API provides the same benefits to internal components:

- Components level triggered instead of edge triggered -- no “missing events” issues.

Resulting in a Simpler, more robust system that can easily recover from failure of components.

- No single point of failure.
- Simple master components.

Why No hidden internal APIs?

Also makes Kubernetes composable and extensible.

- Default component not working for you?
  - Turn it off and replace it with your own.
- Additional functionality not yet available?
  - Write your own and to add it.

Kube API Data

Kubernetes API has lots of data that is interesting to workloads

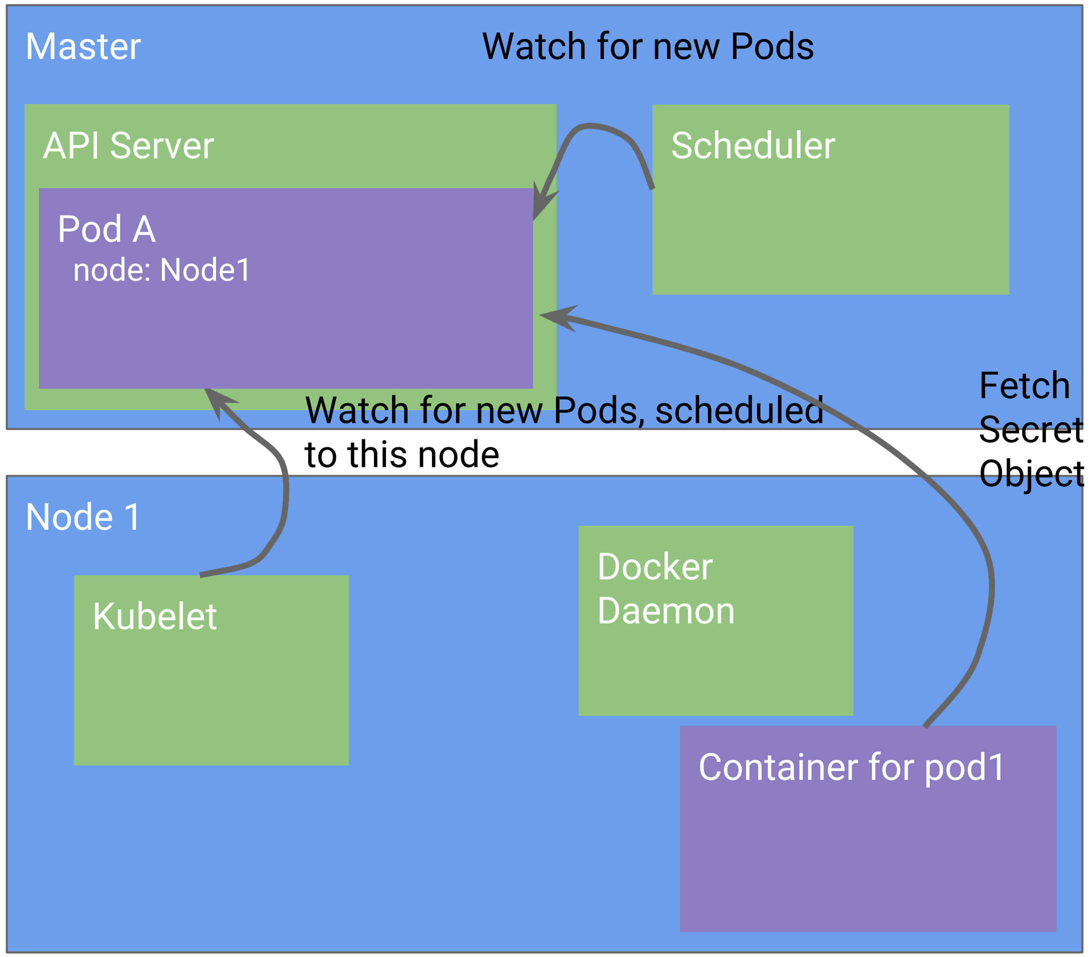
- Secrets - Sensitive info stored in KubeAPI
  - e.g. passwords, certificates, etc.
- ConfigMap - Configuration info stored in KubeAPI
  - e.g. application startup parameters, etc.
- DownwardAPI - Pod information in KubeAPI
  - e.g. name/namespace/uid of my current pod.

Fetching Kube API Data

How does application fetch secrets, config map, etc. information?

Principle: No hidden internal APIs.

Obvious solution: Modify app to read directly from API Server.



Principle #3

Meet the user where they are.

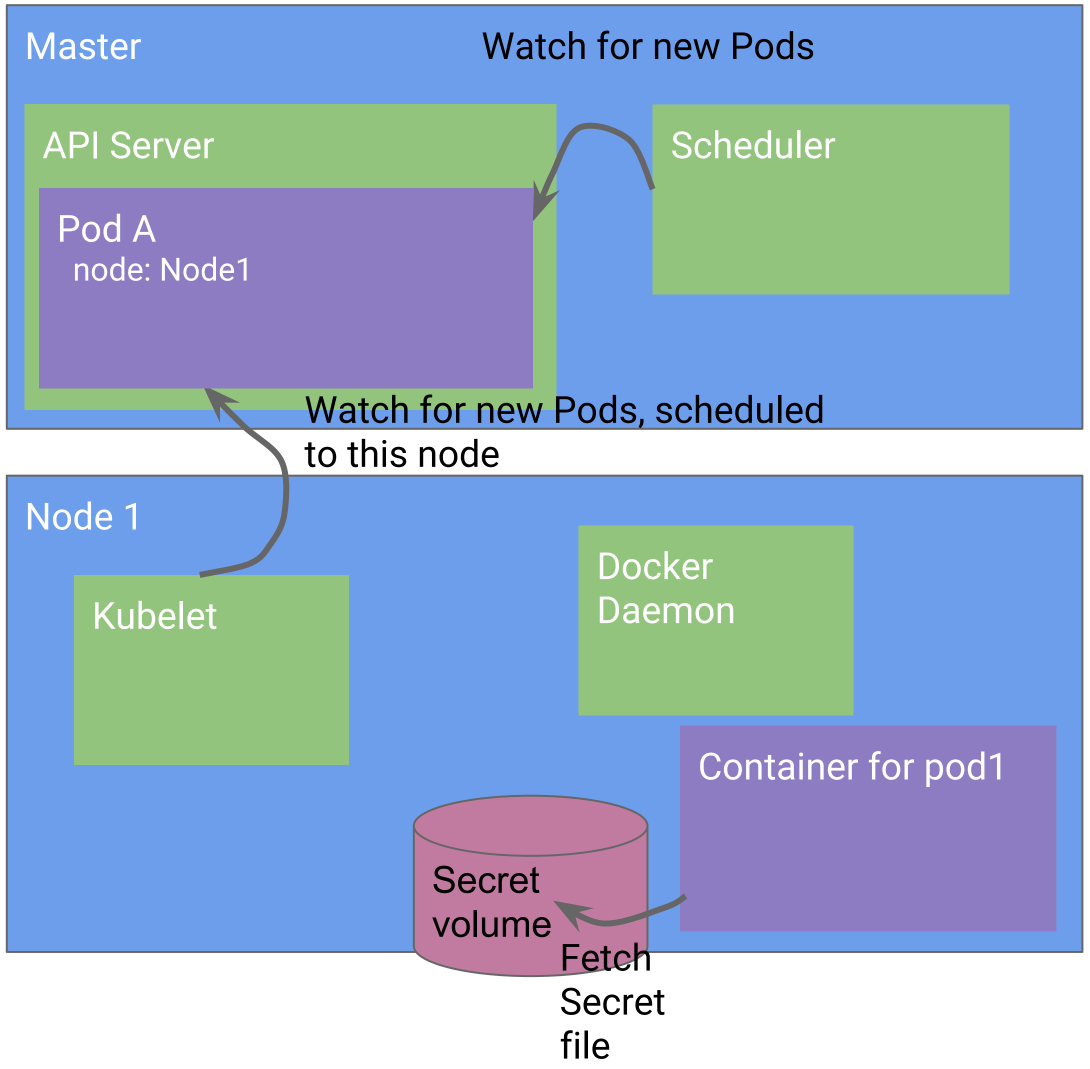
Meet the user where they are.

Before:

- App must be modified to be Kubernetes aware.

After:

- If app can load config or secret data from file or environment variables it doesn't need to be modified!



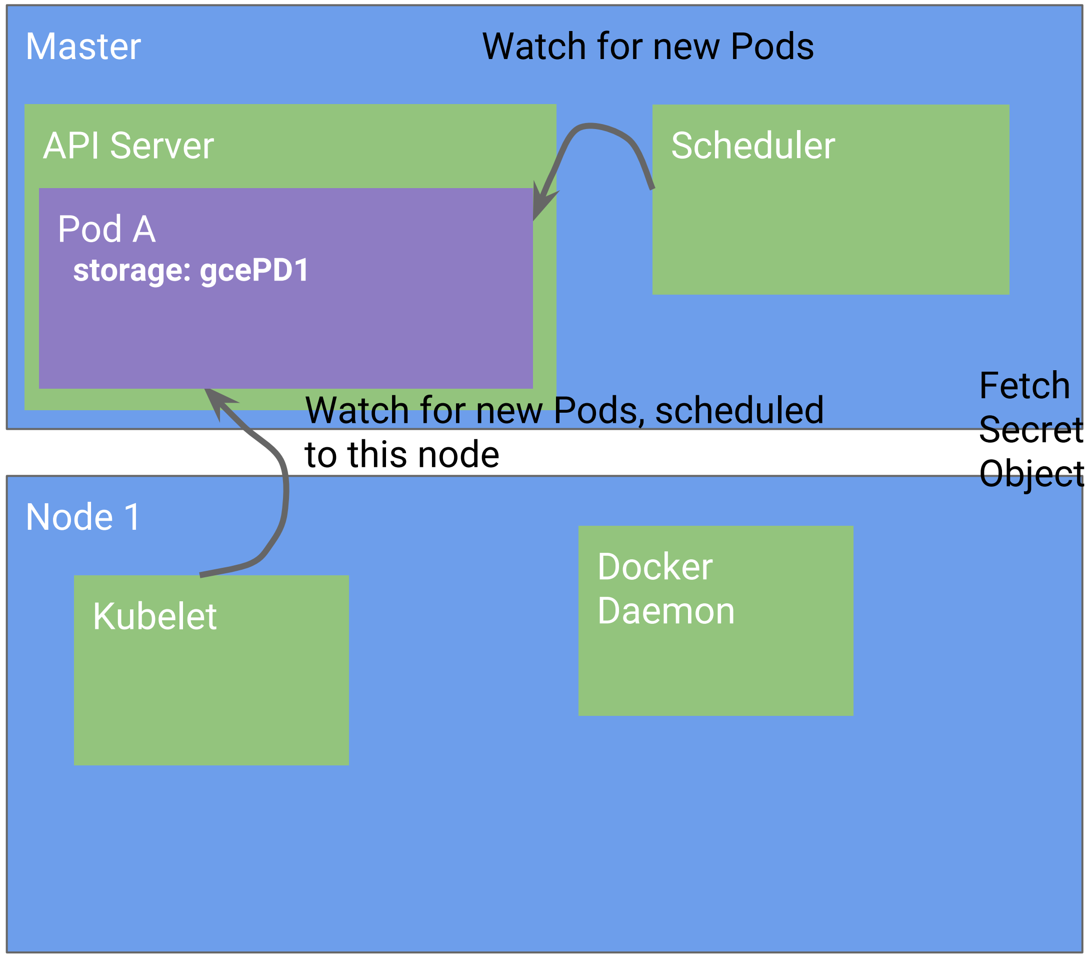
Why meet the user where they are?

Minimize hurdles for deploying workloads on Kubernetes.

Increases adoption.

Remote Storage

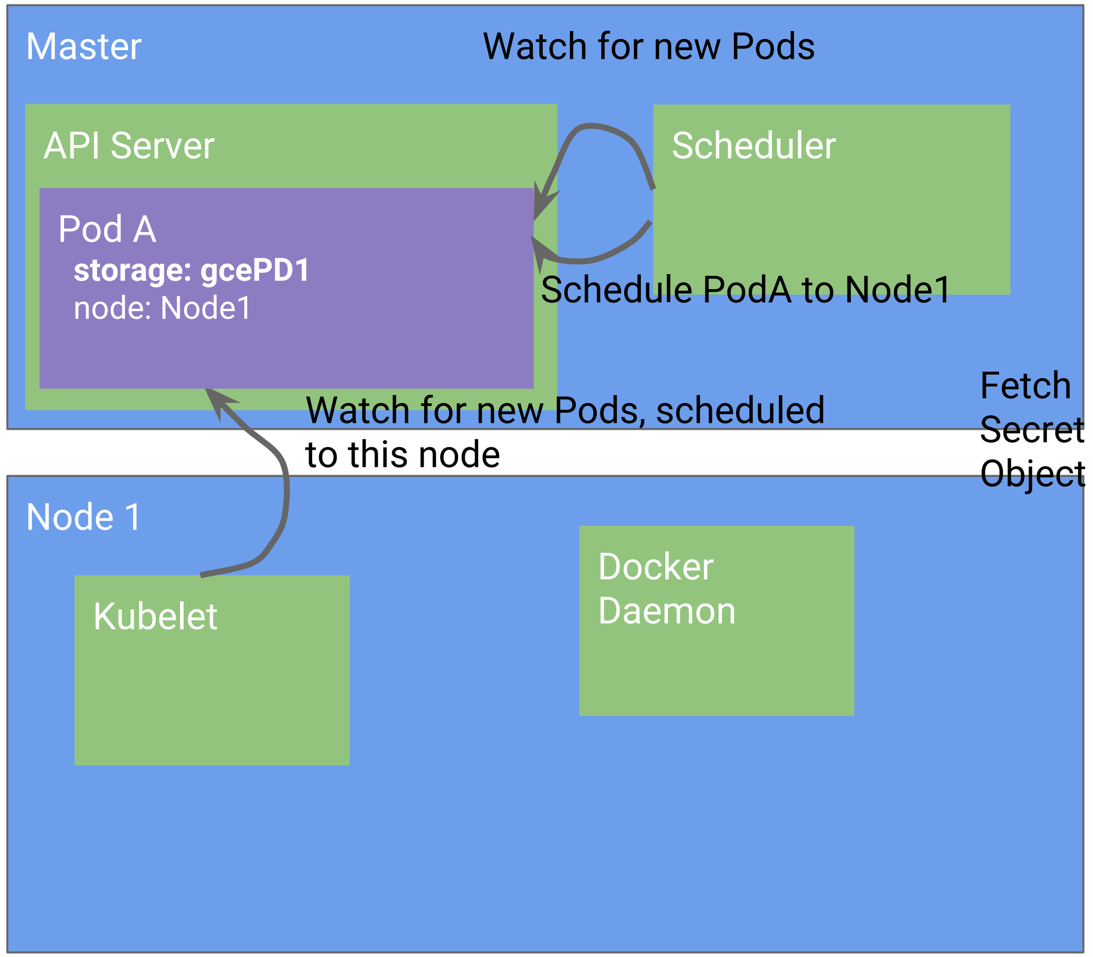
Could directly reference a remote volume (GCE PD, AWS EBS, NFS, etc.) in pod definition.



Remote Storage

Could directly reference a remote volume (GCE PD, AWS EBS, NFS, etc.) in pod definition.

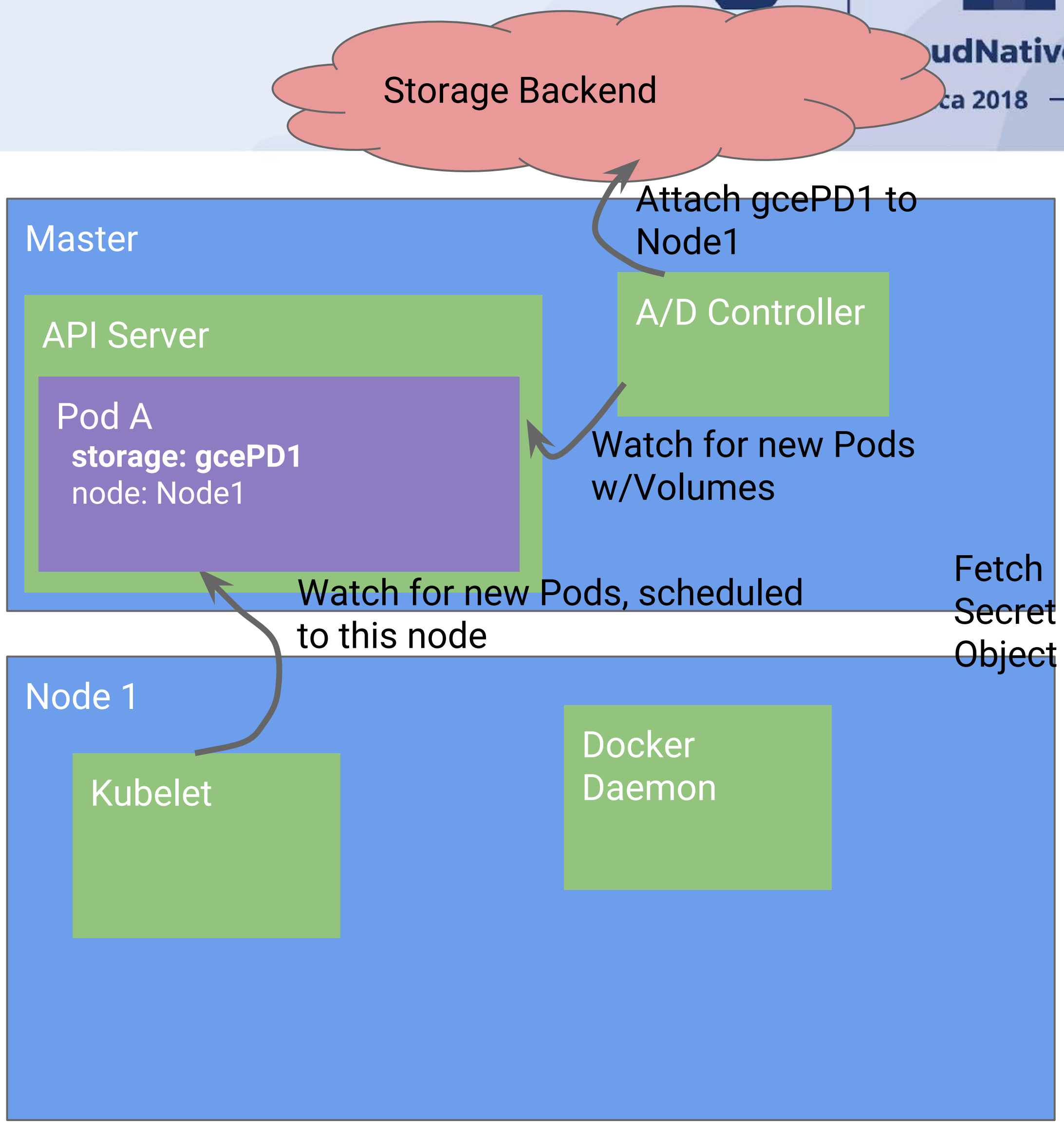
Kubernetes will automatically make it available to workload



Remote Storage

Could directly reference a remote volume (GCE PD, AWS EBS, NFS, etc.) in pod definition.

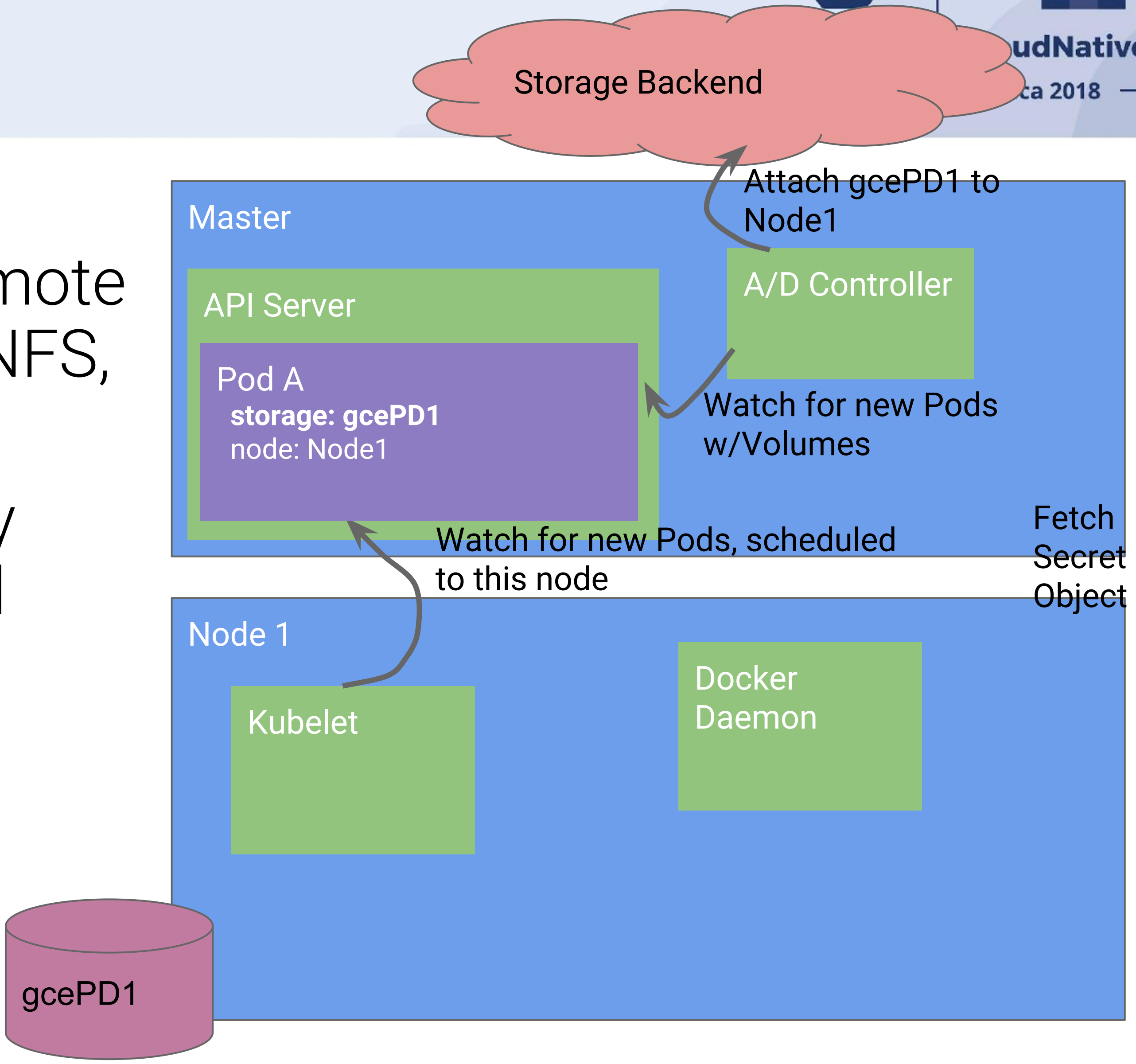
Kubernetes will automatically make it available to workload



Remote Storage

Could directly reference a remote volume (GCE PD, AWS EBS, NFS, etc.) in pod definition.

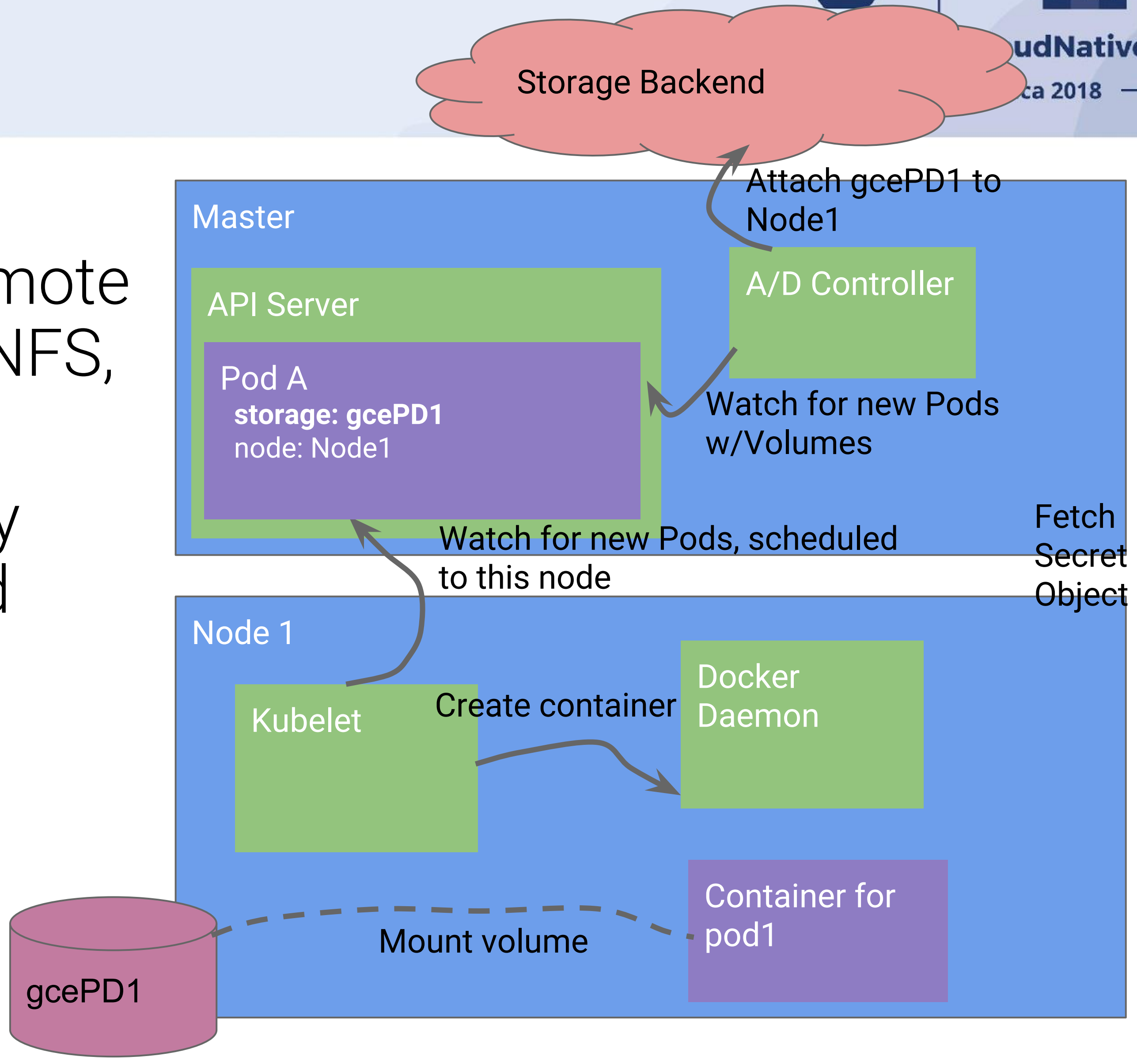
Kubernetes will automatically make it available to workload



Remote Storage

Could directly reference a remote volume (GCE PD, AWS EBS, NFS, etc.) in pod definition.

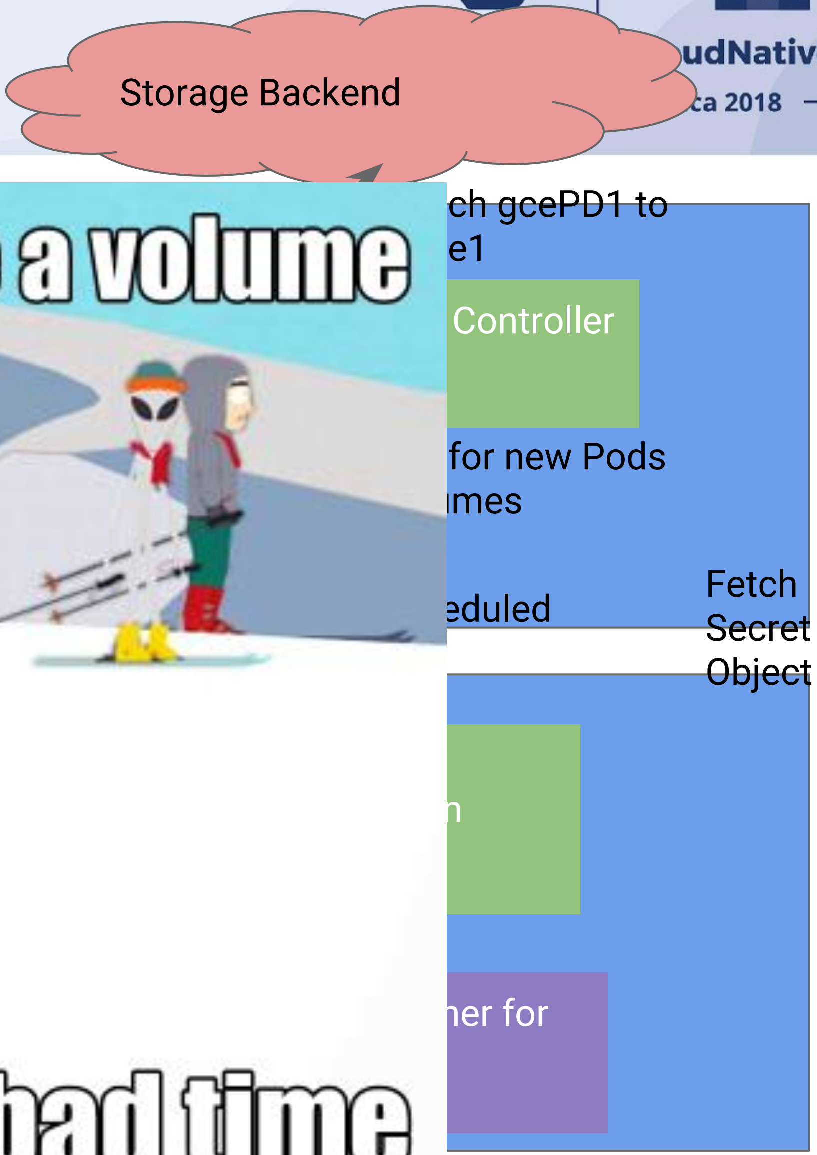
Kubernetes will automatically make it available to workload

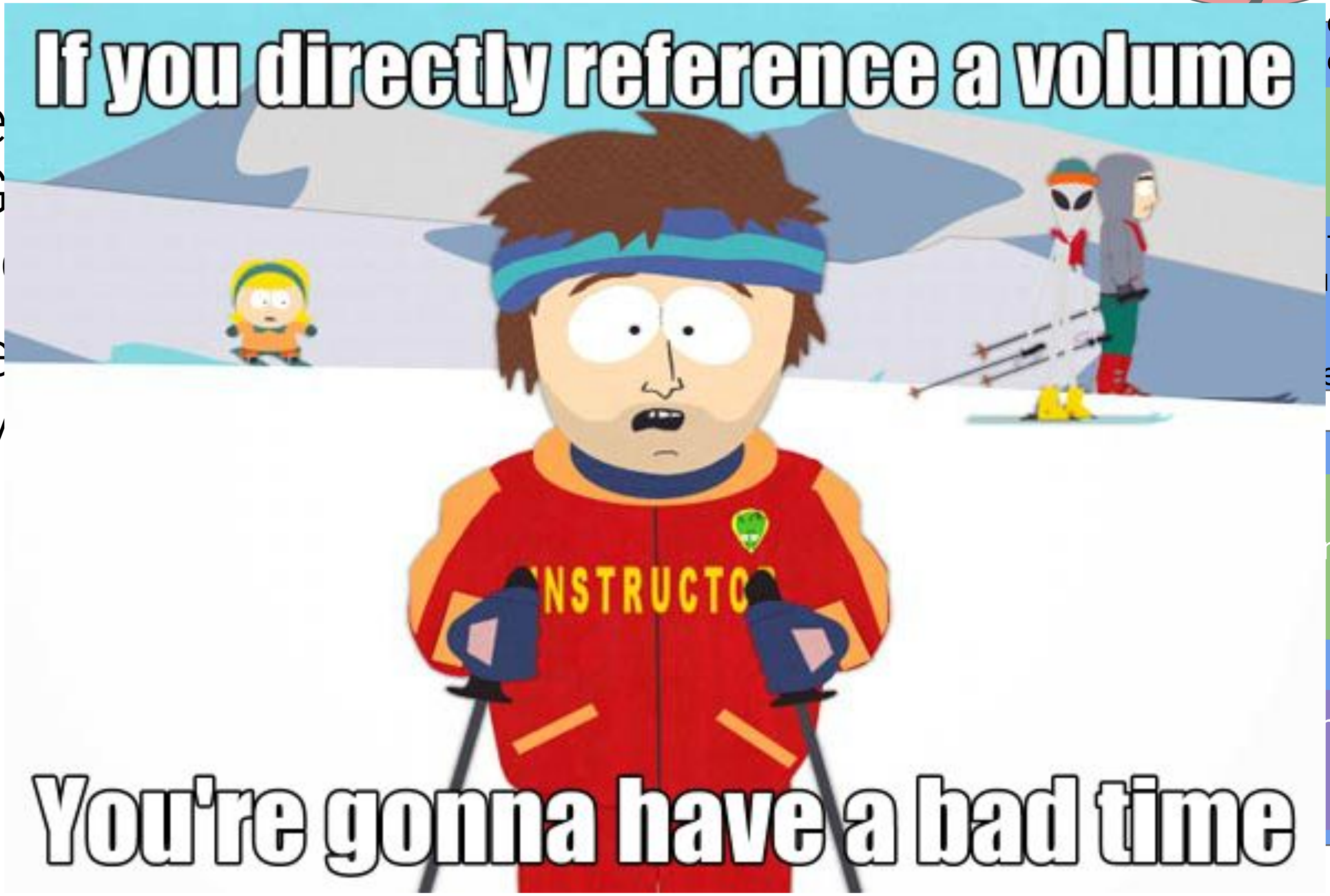


Remote Storage

Could dire
volume (G
etc.) in po

Kubernete
make it av





If you directly reference a volume
NSTRUCTO
You're gonna have a bad time

Principle #4

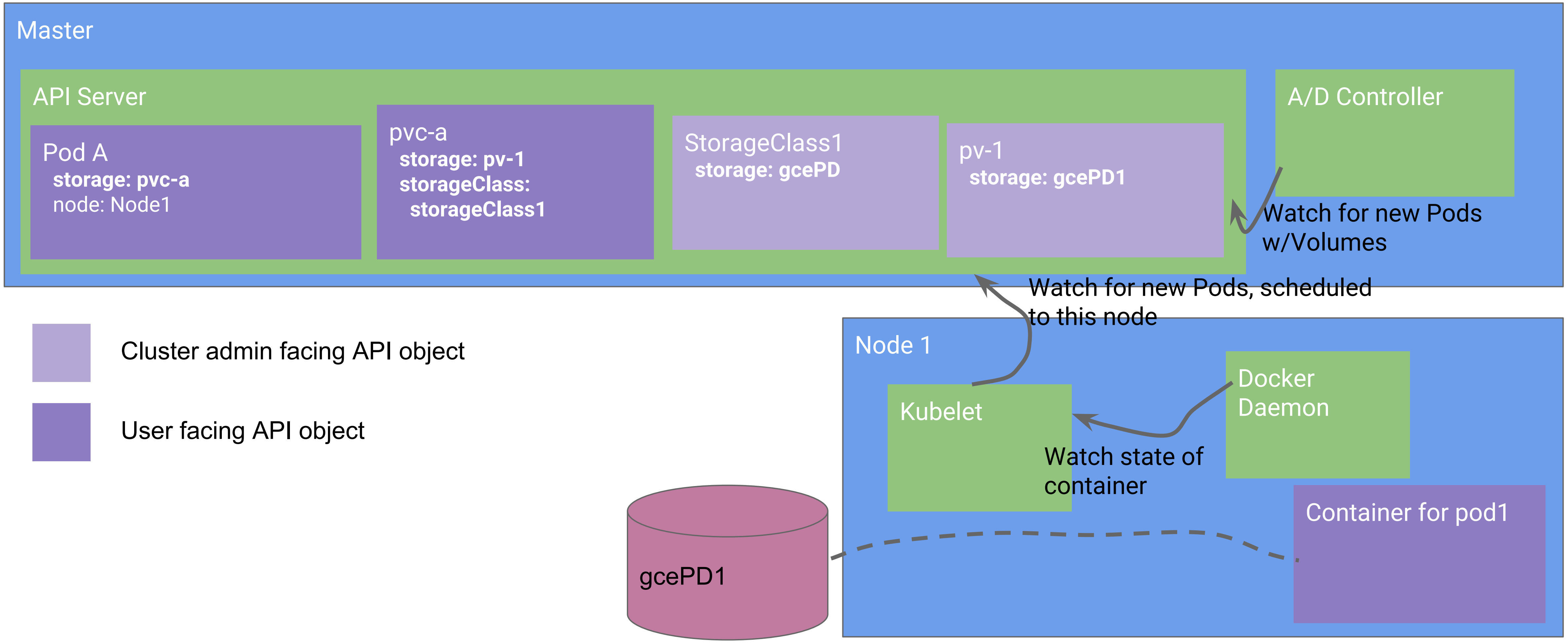
Workload portability

PVC/PV

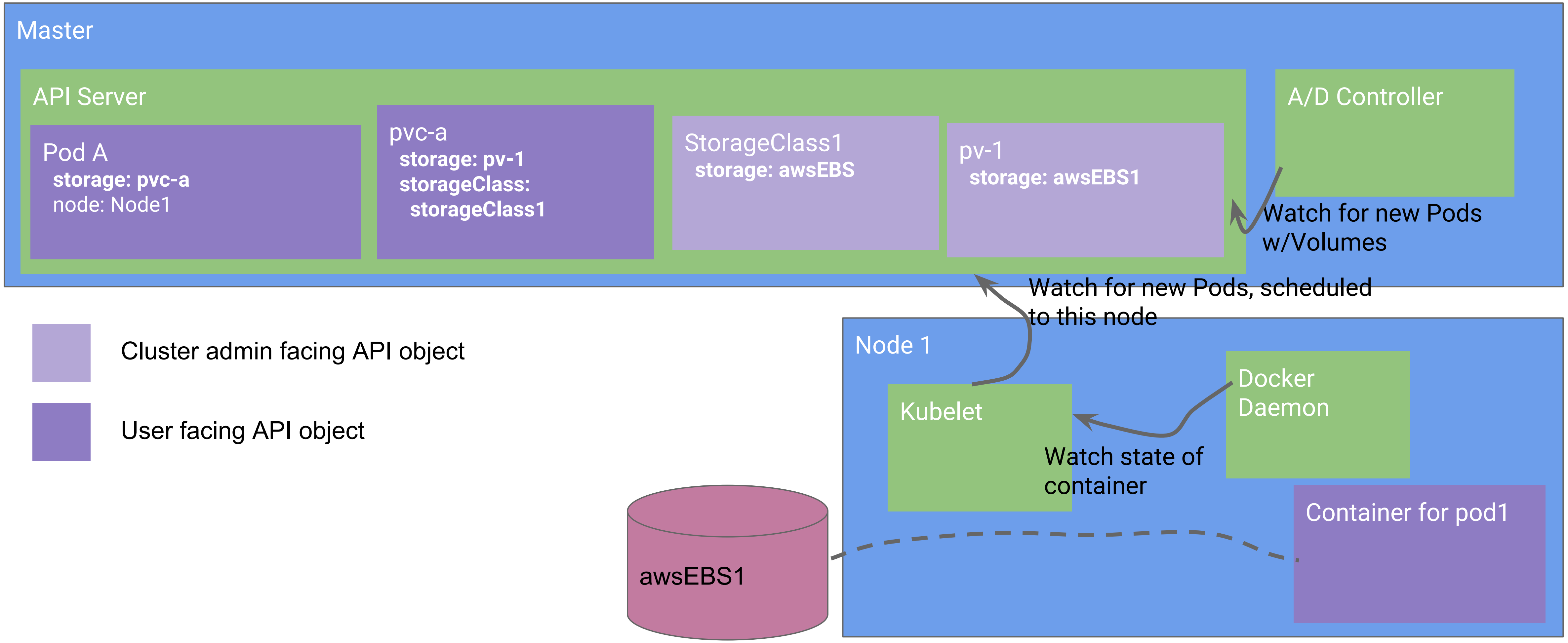
PersistentVolume and PersistentVolumeClaim Abstraction

Decouple storage implementation from storage consumption

PVC/PV


Master
API Server
Pod A
storage: pvc-a
node: Node1
pvc-a
storage: pv-1
storageClass:
storageClass1
StorageClass1
storage: gcePD
pv-1
storage: gcePD1
A/D Controller
Watch for new Pods w/Volumes
Watch for new Pods, scheduled to this node
Node 1
Kubelet
Docker Daemon
Watch state of container
Container for pod1
gcePD1
Cluster admin facing API object
User facing API object

PVC/PV



▨	Cluster admin facing API object
■	User facing API object

Why Workload Portability?

Decouple distributed system application development from cluster implementation.

Make Kubernetes a true abstraction layer, like an OS.

Kubernetes Principles Introduced

1. Kube API declarative over imperative.
2. No hidden internal APIs
3. Meet the user where they are
4. Workload portability

KubeCon | CloudNativeCon

North America 2018

Saad Ali
github.com/saad-ali
twitter.com/the_saad_ali