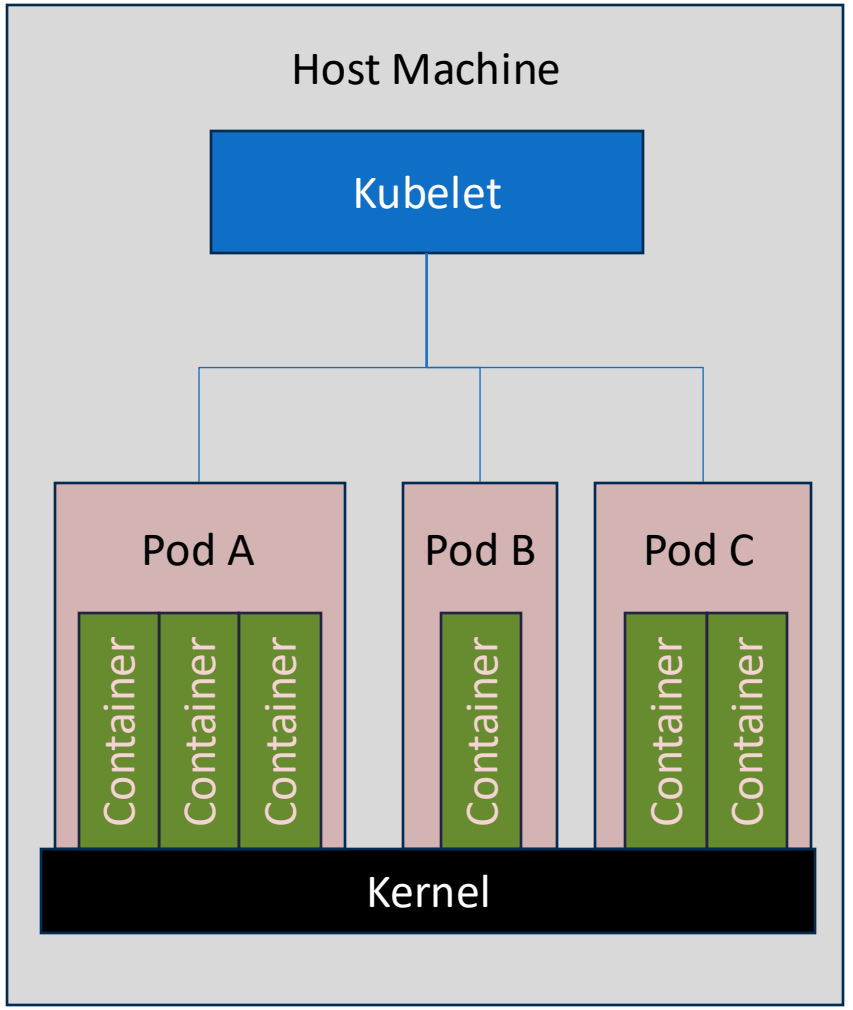
Pods and Workload Resources

Ahmad Alkhansa (ahmad.alkhansa@cnaf.infn.it)

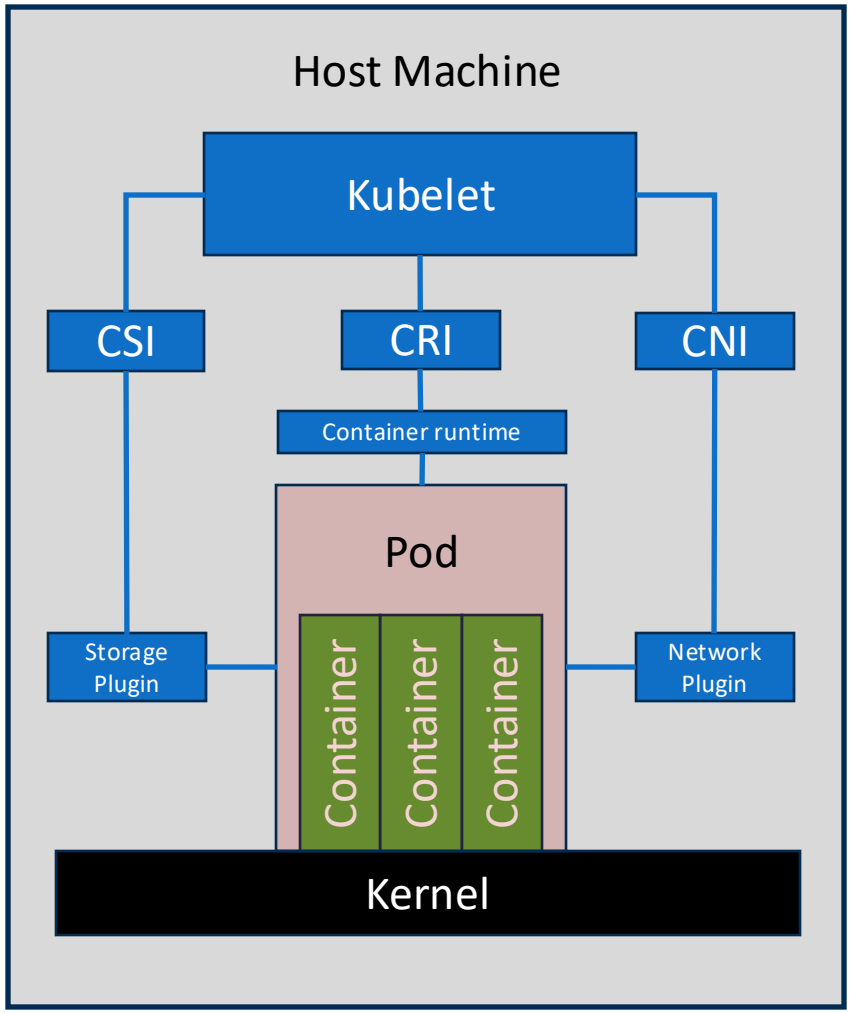
Pod

- A single or group of containers.
- Shared storage and network resources.
- A set of Linux namespaces, cgroups and other isolation contexts.
- Managed by Kubelet at the highest level within the machine.



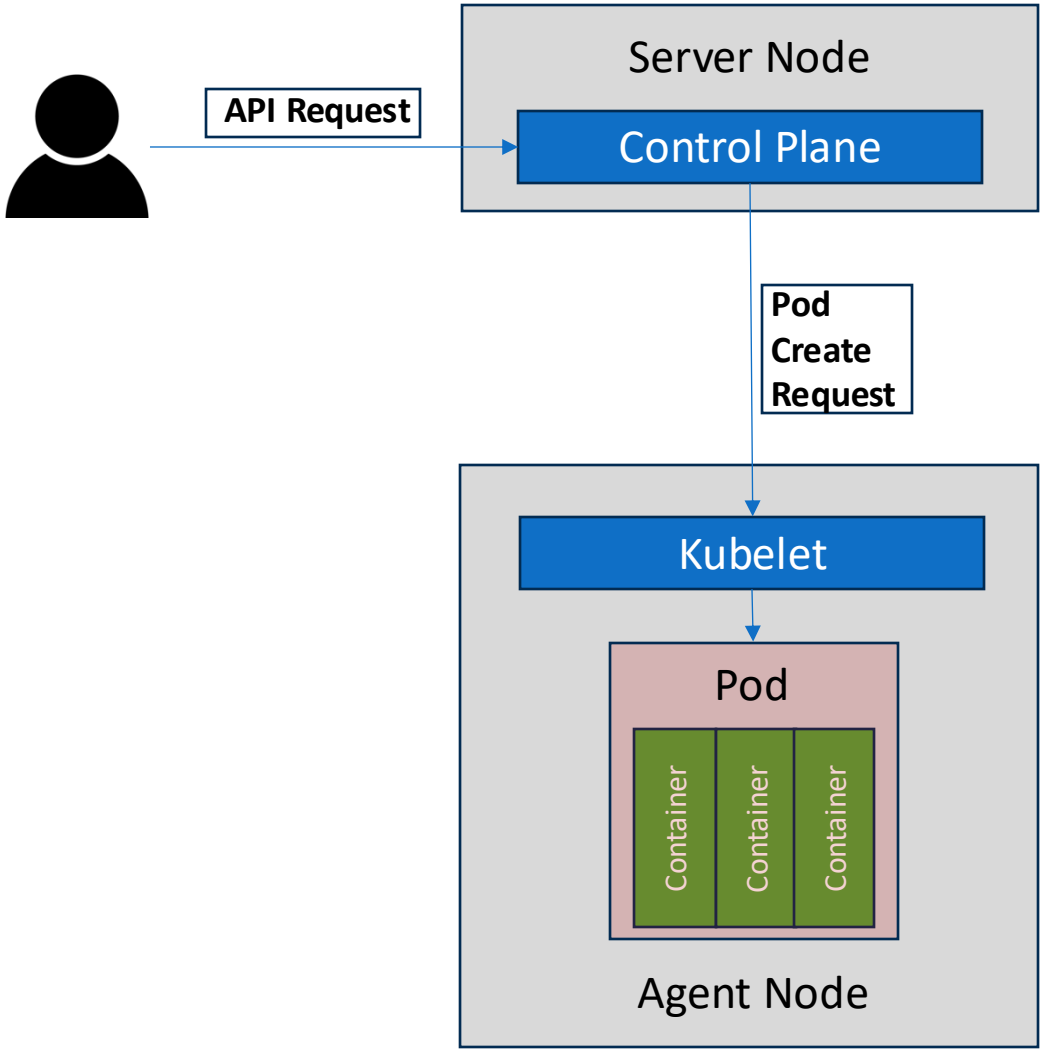
Modular Architecture

- Modular Provisioning Solutions
  - Container Runtime Interface (CRI).
  - Container Network Interface (CNI).
  - Container Storage Interface (CSI).
- Plethra of Solutions
  - Containerd, CRI-O, etc. as container runtime.
  - Calico, Cilium, etc. as networking solution.
  - Longhorn, Ceph, etc. as storage solution.



Workflow

- Client performs an API request containing a YAML manifest.
- Control Plane schedules the Pod and sends a create request to the Kubelet.
- Kubelet communicates with the Interfaces (CRI, CNI & CSI) to provision resources.
- Pod is spawned with its containers.



YAML

- Data serialization Language.
- SuperSet of Json.
- Relies on indentation to represent the data structure.
- Quotation marks are mostly not needed.

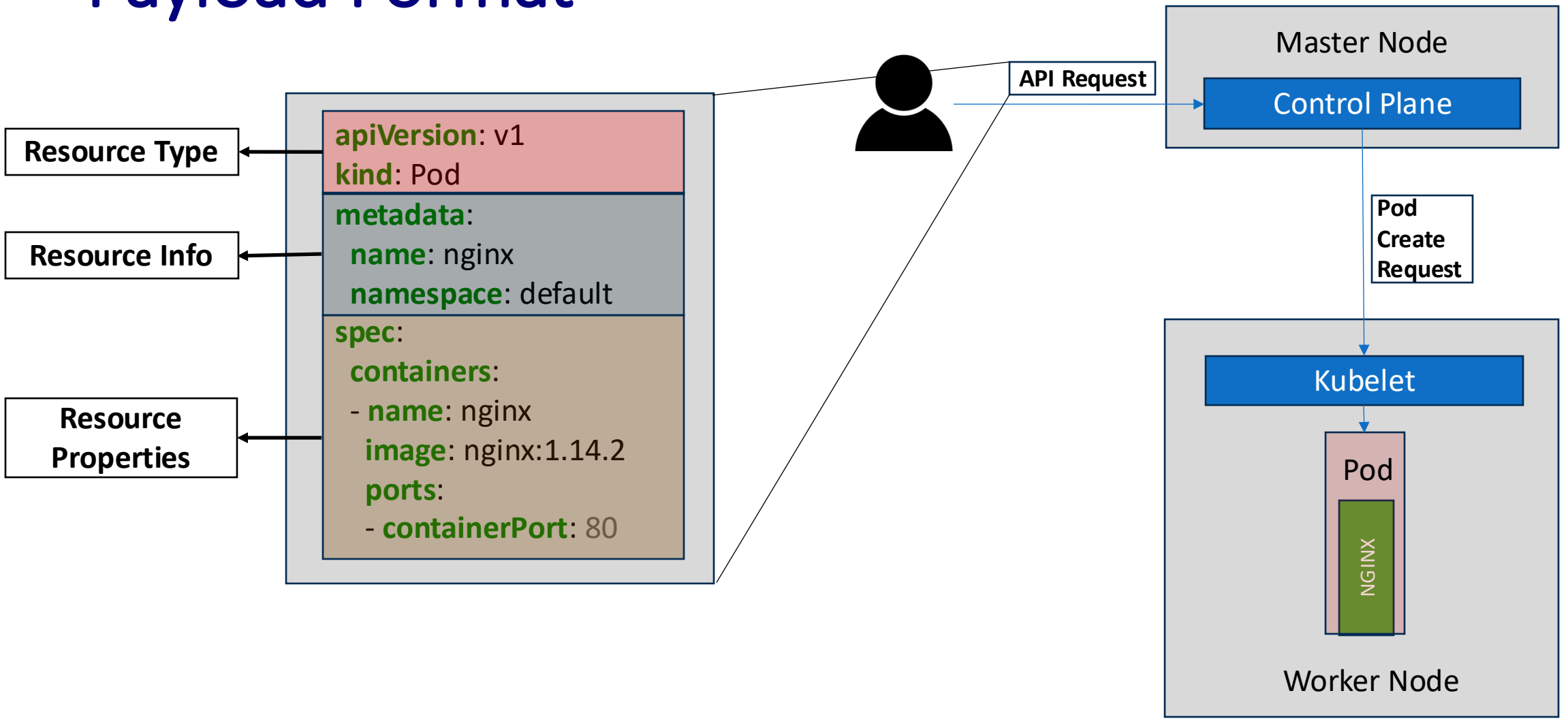
```
{
        "cluster1" : [
                "node1",
                "node2",
                "node3"
        ],
        "cluster2": [
                "nodeA",
                "nodeB",
                "nodeC"
        ]
}
```

```
cluster1:
  - node1
  - node2
  - node3
cluster2:
  - nodeA
  - nodeB
  - nodeC
```

```
{
        "cluster1" : [
                "node1": {
                        "pods": [
                                "app-1A",
                                "app-1B",
                                "app-1C"
                        ],
                        "volumes": [
                                "volume-1A",
                                "volume-1B",
                                "volume-1C"
                        ]
                },
                "node2": {
                        "pods": [
                                "app-2A",
                                "app-2B"
                        ],
                        "volumes": [
                                "volume-2B"
                        ]
                }
        ]
}
```

```
cluster1:
  - node1:
      pods:
      - app-1A
      - app-1B
      - app-1C
      volumes:
      - volume-1A
      - volume-1B
      - volume-1C
  - node2:
      pods:
      - app-2A
      - app-2B
      volumes:
      - volume-2B
```

Payload Format



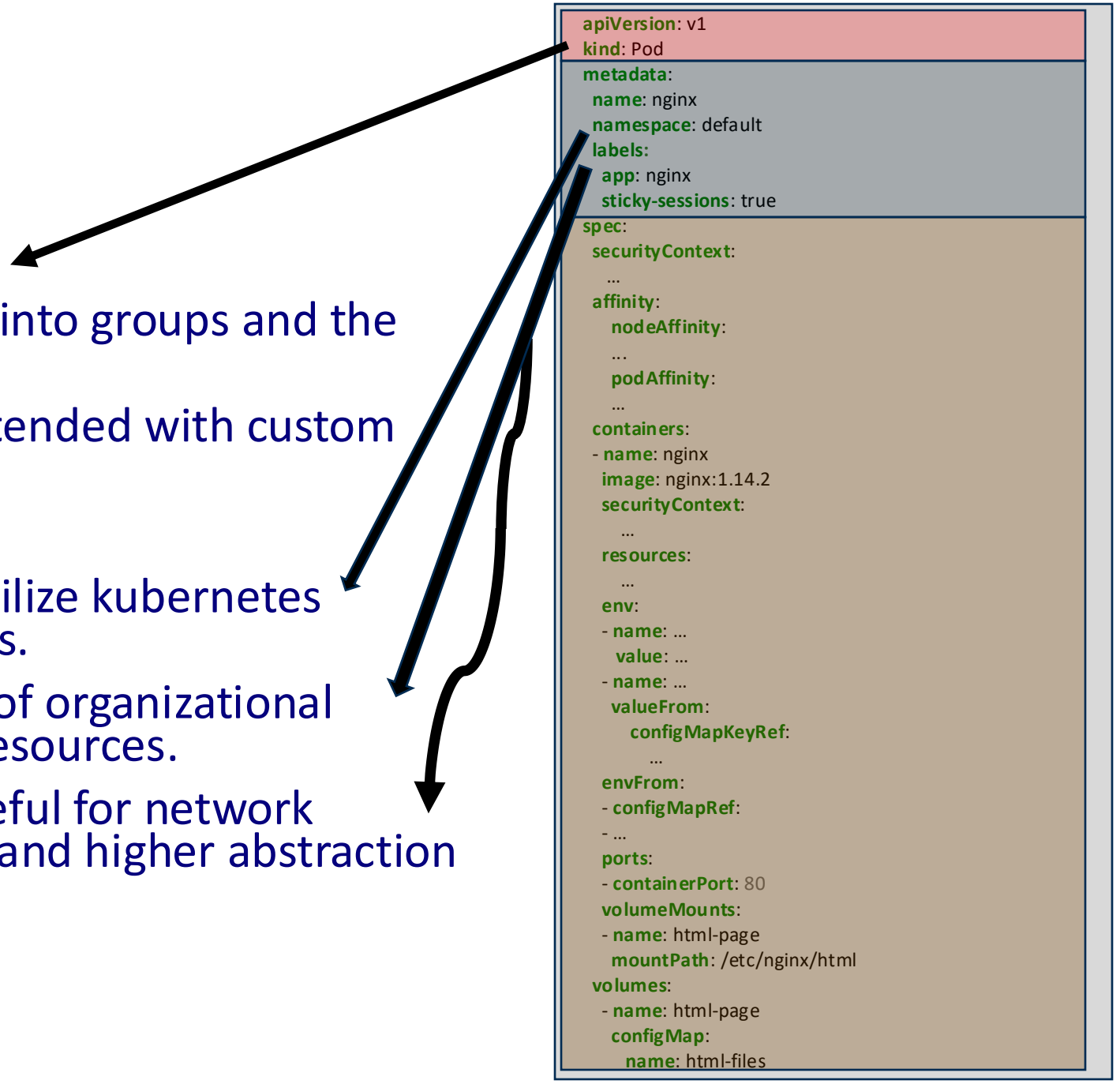
Data Structure

apiVersion & kind:

- Kubernetes API is divided into groups and the pod is at the core level.
- Kubernetes API can be extended with custom resources.

Metadata:

- Namespace compartmentilize kubernetes resources and abstractions.
- Labels allow the addition of organizational information into system resources.
- Labels within pods are useful for network configuration, scheduling and higher abstraction functions.



Data Structure

Affinities Pod Level:

- Affinities/Antiaffinities to different nodes and pods can be defined.
- Complex expressions that control the labels of nodes and pods.

Secutiy Context Pod & Container Level

- Define user, group and other access level for containers within a pod.
- Some parameters may be exclusive to a certain context.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
    sticky-sessions: true
spec:
  affinity:
    nodeAffinity:
      ...
    podAffinity:
      ...
  securityContext:
    ...
  containers:
  - name: nginx
    image: nginx:1.14.2
    securityContext:
      ...
    resources:
      ...
    env:
    - name: ...
      value: ...
    - name: ...
      valueFrom:
        configMapKeyRef:
          ...
    envFrom:
    - configMapRef:
    - ...
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html-page
      mountPath: /etc/nginx/html
  volumes:
    - name: html-page
      configMap:
        name: html-files
```

Data Structure

Resources Container Level:

- Defines container's minimum CPU and memroy request and their maximum limits.

Environment Variables Container Level:

- Environment variables can be defined within the specifications block.
- Referencing Configmaps and Secrets is possible.

Data Structure

Volume Mounts Container Level:

- Defines the mount path and access mode to the specified volume.

Volumes Pod Level:

- Different types of storage volumes can be specified.
- Out-of-the-box solutions can be emptydir, configmaps, secrets and host path.
- Different integrations with external storage exist and can be mounting using persistent volume claims.

ConfigMaps and Secrets

- Decouple application configuration from its code.
- Key-value data structures.
- Stored within the control plane database.
- Configmaps are intended for non-confidential data.
- Secrets are used to store credentials, private keys, x509 certificates, etc.

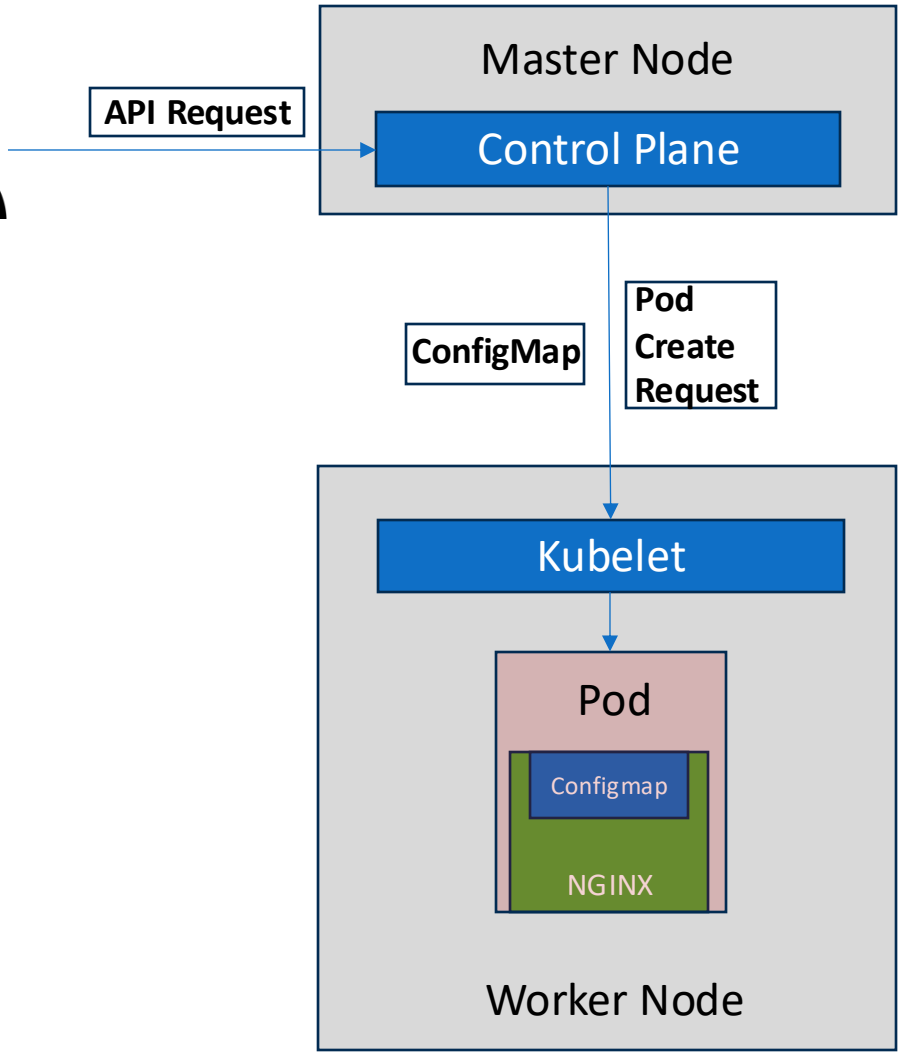
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: html-files
  namespace: default
data:
  index.html: |
    <!DOCTYPE html>
    <html>
    <head>
    <title>Say Hi</title>
    </head>
    <body>
    Hello World!
    </body>
    </html>
```

```
apiVersion: v1
kind: Secret
metadata:
  name: docker-pull-creds
  namespace: default
type: kubernetes.io/dockerconfigjson
stringData:
  .dockerconfigjson: |
    {
        "auths": {
          "https://registry": {
              "auth": “base64 encode”
          }
        }
    }
```

ConfigMaps and Secrets

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
    sticky-sessions: true
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html-page
      mountPath: /usr/share/nginx/html
  volumes:
    - name: html-page
      configMap:
        name: html-files
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: html-files
  namespace: default
data:
  index.html: |
    <!DOCTYPE html>
    <html>
    <head>
    <title>Say Hi</title>
    </head>
    <body>
    Hello World!
    </body>
    </html>
```



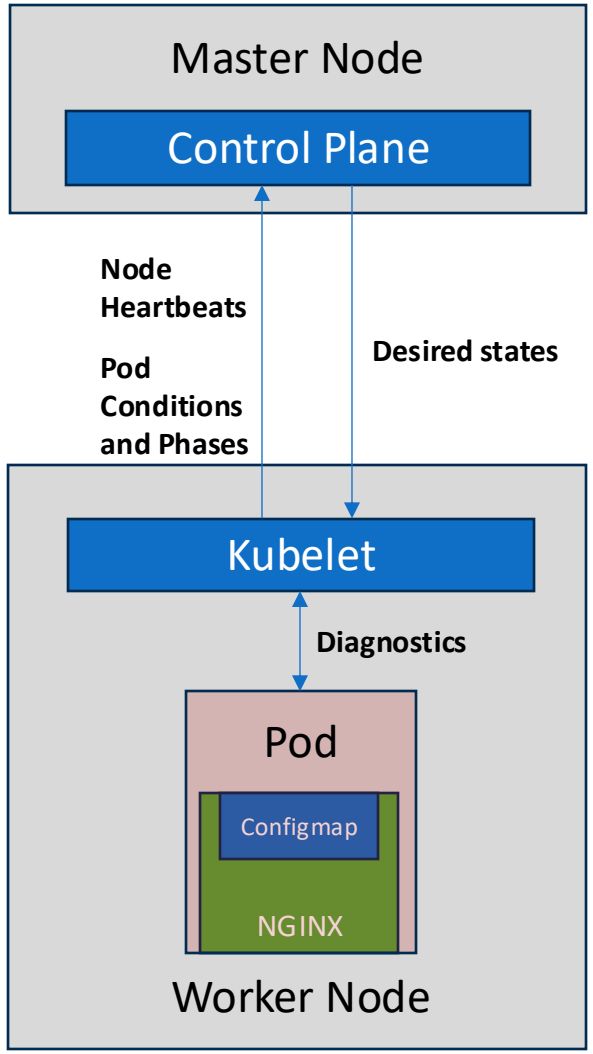
Pod Lifecycle

Node Level

- The control plane automatically reschudale a pod after the node failure.
- There's a grace period before determining node failure.

Pod Level

- Phases report the current state of a pod that can be **Pending, Running, Succeeded, Failed, Unknown**.
- Conditions show **PodScheduled, PodReadyToStartContainers, ContainersReady, Initialized, Ready**.



Pod Lifecycle

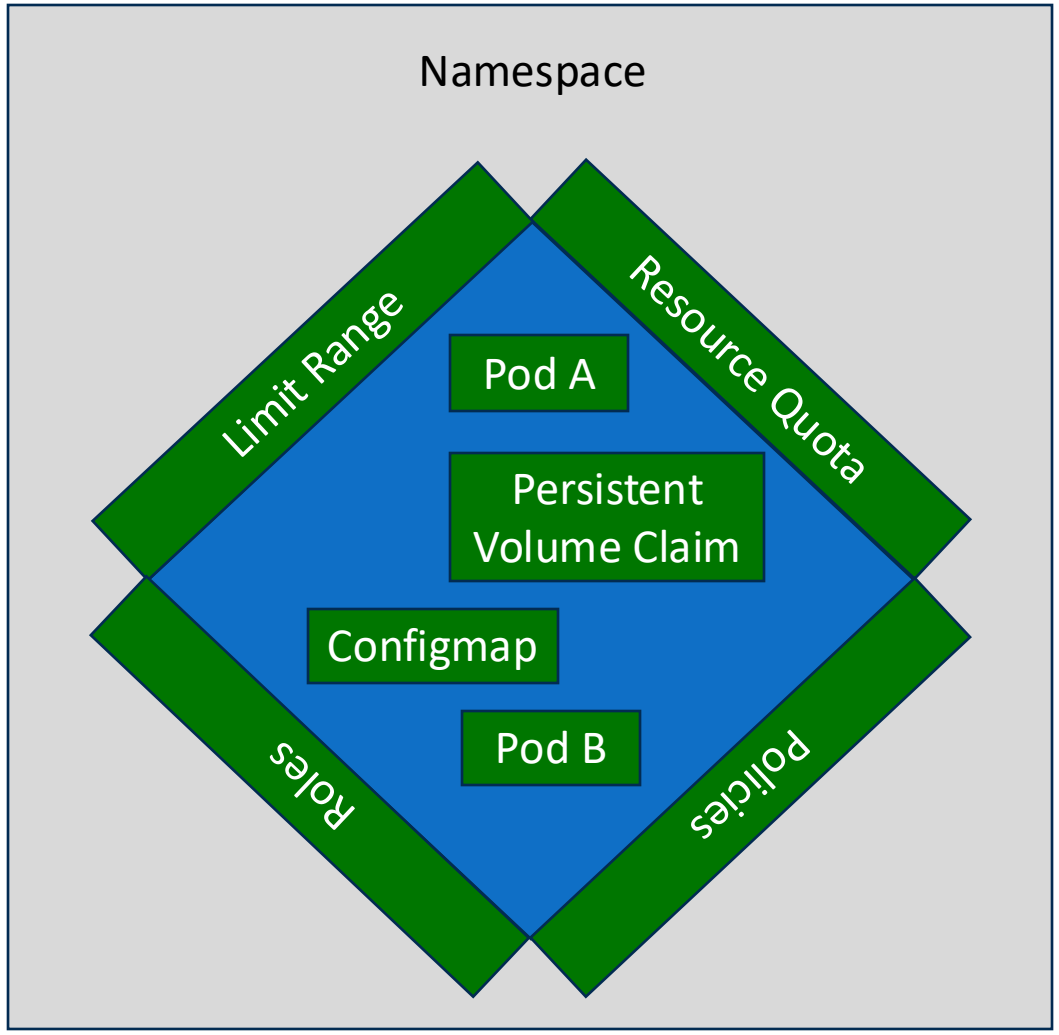
Container Level

- Containers have 3 states **Waiting, Running and Terminated**.
- **Liveness, Readiness and Startup Probes** can be specified to instruct the kubelet to perform checks.
- Probe mechanisms can be **exec, grpc, httpGet and tcpSocket**.
- **PostStart and PreStop** can be determined with the same mechanisms.
- **Init and side car containers** start before the main container.

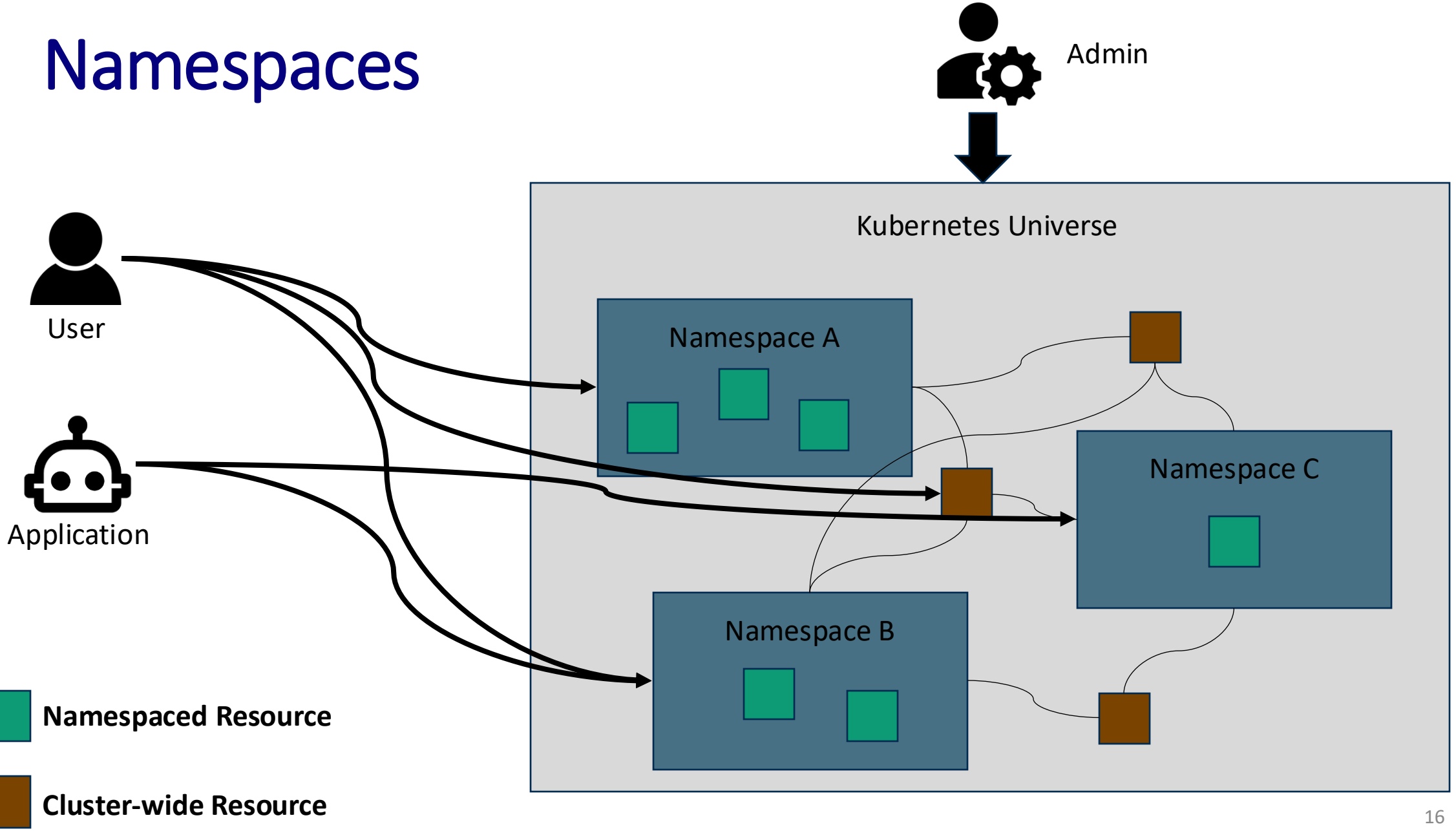
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
    sticky-sessions: true
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
    restartPolicy: Always
    readinessProbe:
      httpGet:
        path: /
        port: 80
    volumeMounts:
    - name: html-page
      mountPath: /etc/nginx/html
  volumes:
    - name: html-page
      configMap:
        name: html-files
```

Namespaces

- Environments to organize and define **access to cluster resources**.
- **Access Control** can be handled using roles and attributes.
- CPU, memory and storage namespace limits can be defined using **Resource Quotas**.
- **Limit ranges** are used to define limits at the level of single pod/container.

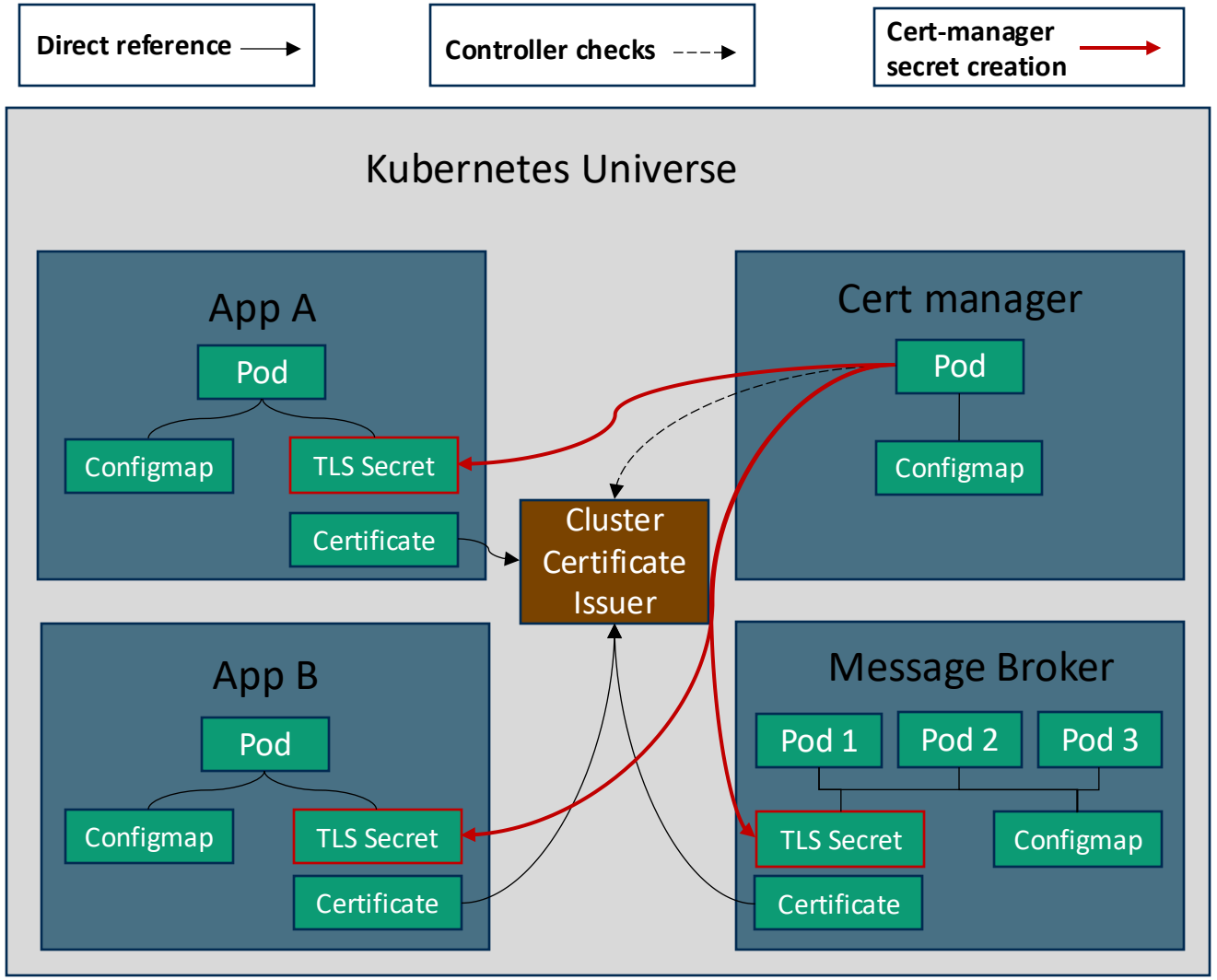


Namespaces


Admin
Kubernetes Universe
User
Application
Namespace A
Namespace B
Namespace C
Namespaced Resource
Cluster-wide Resource

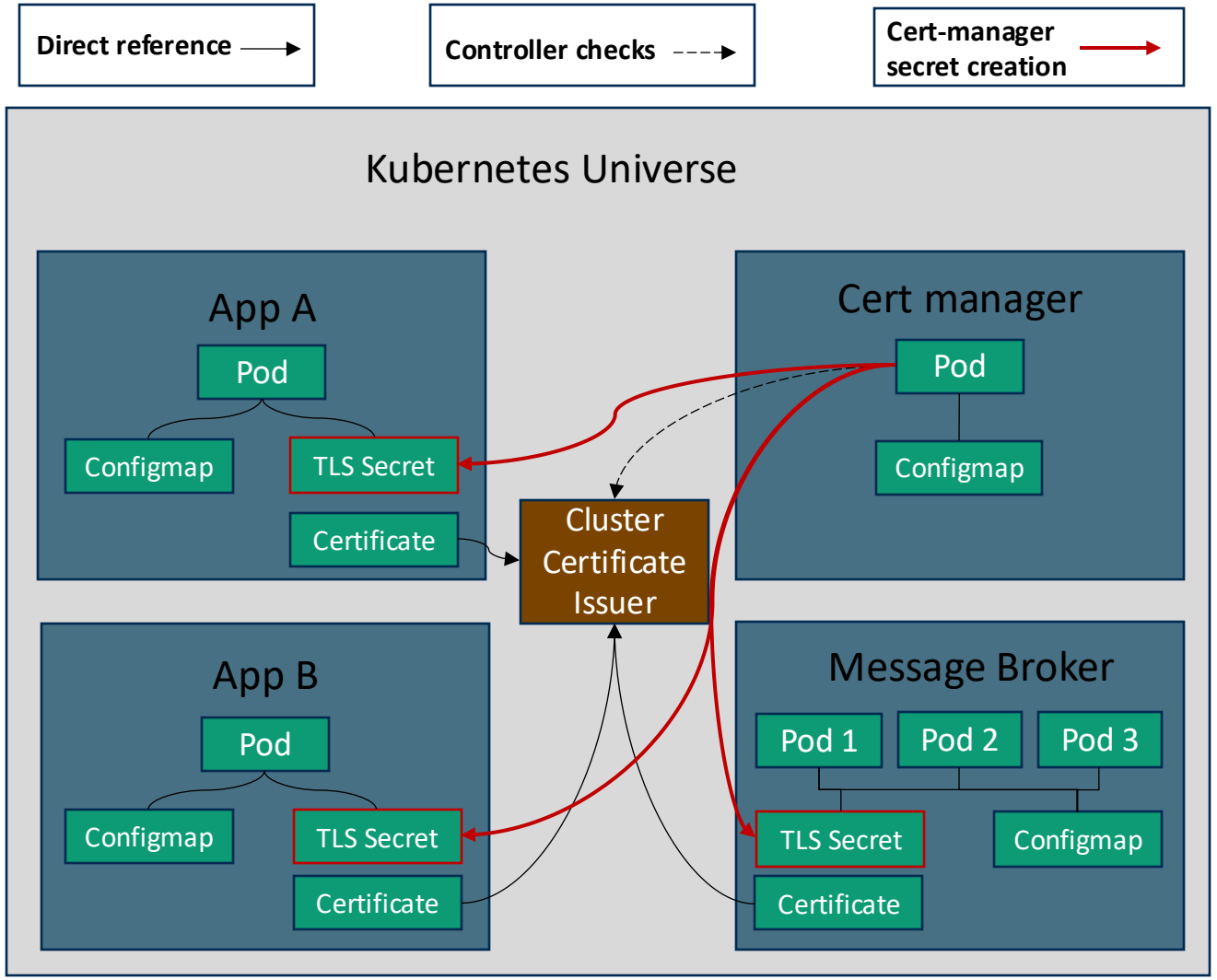
Scenario

- Administrator creates all **namespaces** and defines **resource quotas and access control**.
- Administrator deploys the **Cert Manager application** and creates the **cluster certificate issuer**.
- All teams send requests to the **API server of Kubernetes** to create **certificates** signed by the cluster certificate issuer.
- Cert Manager creates the **certificates** in a form of **TLS secrets**.
- Apps A and B use the **certificates** to **authenticate** with the message broker.



Important Notes

- API of Kubernetes can be **Extended** with **custom resource definitions**.
- Cluster Certificate Issuer and Certificate are **custom resources.**
- Cluster Certificate Issuer and Certificate are **cluster-wide resources**.
- Cert Manager is a **Kubernetes controller**.
- Cert Manager has a **service account** with **access level** defined by the administrator.

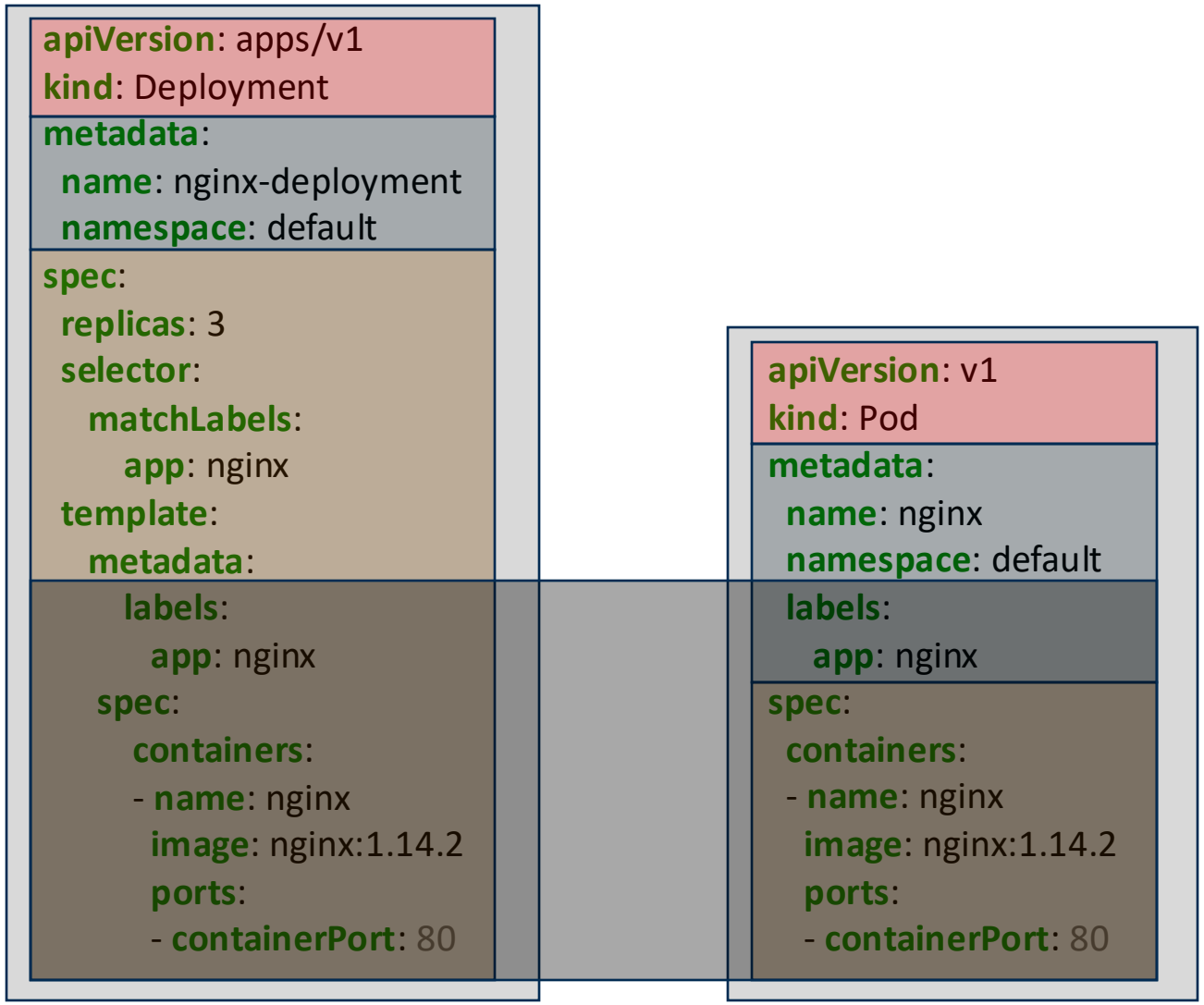


Workload Resources

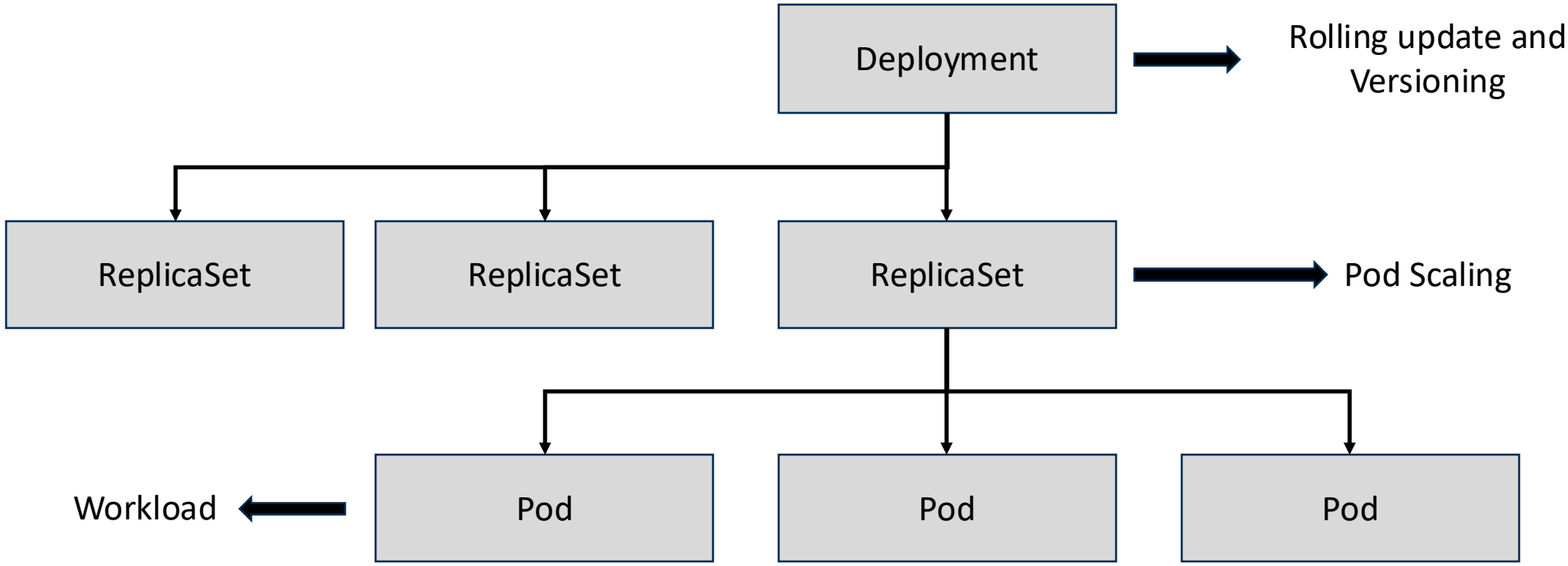
- Different **abstractions** that define the **statefuleness, distribution and nature** of the applications running on top of Kubernetes.
- Controllers with different mechanisms to maintain the **desired state of a workload**.
- **Deployment** manages and scales stateless applications.
- **StatefulSets** provides stable network identity and persistent storage, and ordered rolling update.
- **DaemonSets** are meant to exist within every Kubernetes node that meets certain criteria.
- **Job** and **CronJob** are abstractions for executing tasks that run till completion.

Deployment

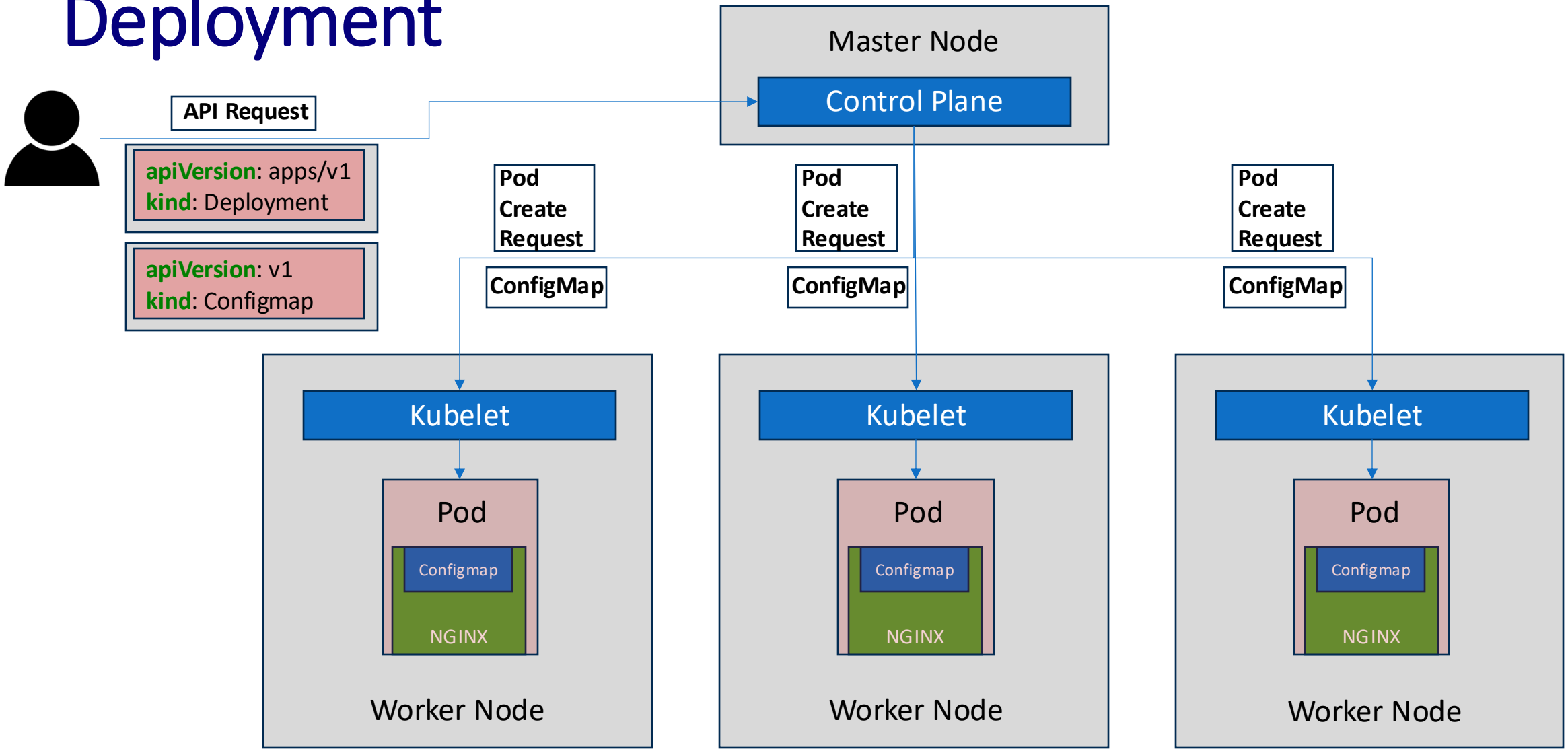
- Based on a pod template.
- Scaling pods to several replicas.
- Stores versions of deployment instances.
- Rollout and rollback features.



Resource Hierarchy And Ownership

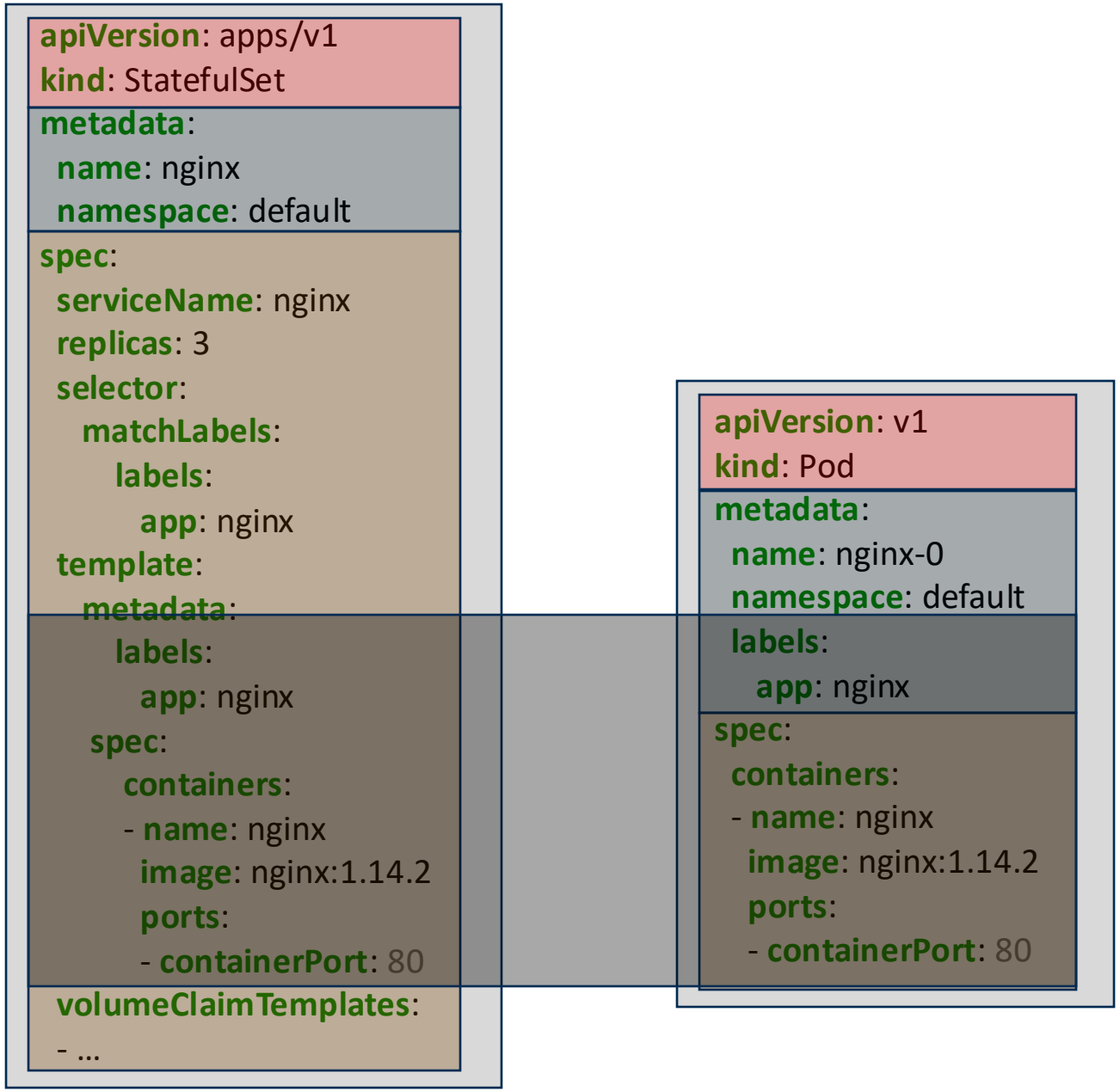


Deployment


Master Node
Control Plane
API Request
apiVersion: apps/v1
kind: Deployment
apiVersion: v1
kind: Configmap
Pod Create Request
ConfigMap
Pod Create Request
ConfigMap
Pod Create Request
ConfigMap
Kubelet
Pod
Configmap
NGINX
Worker Node
Kubelet
Pod
Configmap
NGINX
Worker Node
Kubelet
Pod
Configmap
NGINX
Worker Node

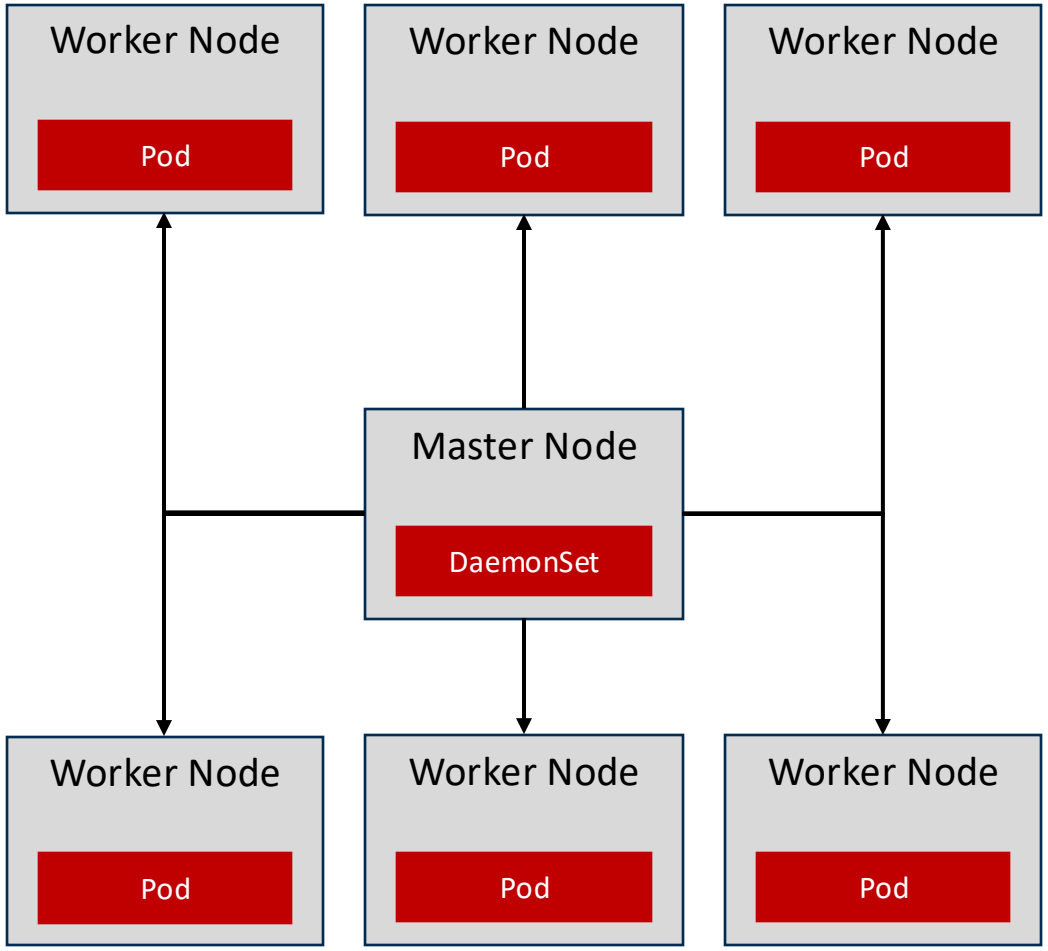
StatefulSet

- Pods' name consist of the **StatefulSet name + ordinal**.
- The pod derives its **hostname from the pod name**.
- Possibility to **reach** a single pod by it's name **from anywhere in the cluster** using **Headless Service**.
- Each **Replica** can be mounted with a **separate persistent volume** claims using **Claim Template**.



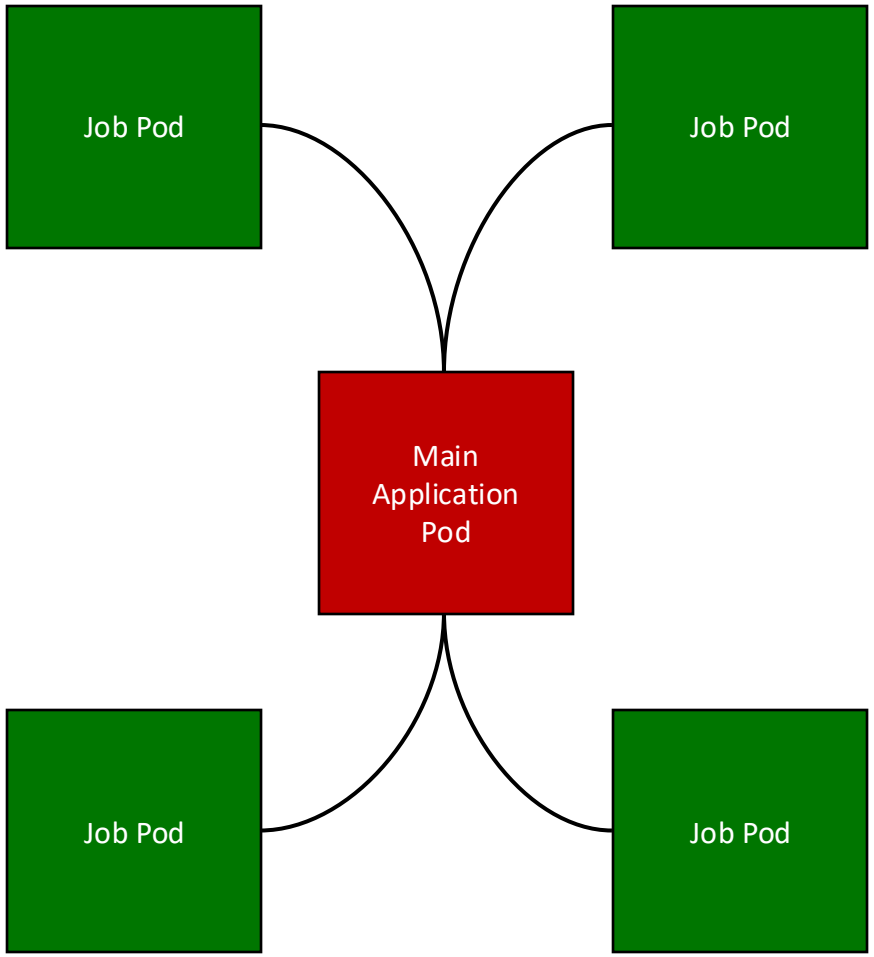
DaemonSet

- Pods deployed within every possible node.
- In case of node affinities and selection, scheduling all matching nodes.
- Possibility for rollout and rollback.
- Commonly used for Node Plugins.



Jobs and CronJobs

- Jobs are containers that run until completion.
- Possibility to create fine-grained pod failure/success policy.
- Ability to define parallelism.
- CronJobs run periodically and use Jobs as templates.



References

- Pod - https://kubernetes.io/docs/concepts/workloads/pods/
- Kuberentes components - https://kubernetes.io/docs/concepts/overview/components/
- CRI - https://kubernetes.io/docs/concepts/architecture/cri/
- CSI - https://kubernetes-csi.github.io/docs/introduction.html
- CNI - https://github.com/containernetworking/cni
- YAML - https://www.ibm.com/topics/yaml
- JSON - https://www.json.org/json-en.html
- ConfigMaps - https://kubernetes.io/docs/concepts/configuration/configmap/
- Secrets - https://kubernetes.io/docs/concepts/configuration/secret/
- Pod Lifecycle - https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

References

- Namespaces - https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
- Cert-manager - https://cert-manager.io/docs/concepts/issuer/
- Custom resources - https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
- Workloads - https://kubernetes.io/docs/concepts/workloads/
- Deployment - https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- ReplicaSet - https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/
- StatefulSet - https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
- DaemonSet - https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/
- Job - https://kubernetes.io/docs/concepts/workloads/controllers/job/
- API reference - https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.24/