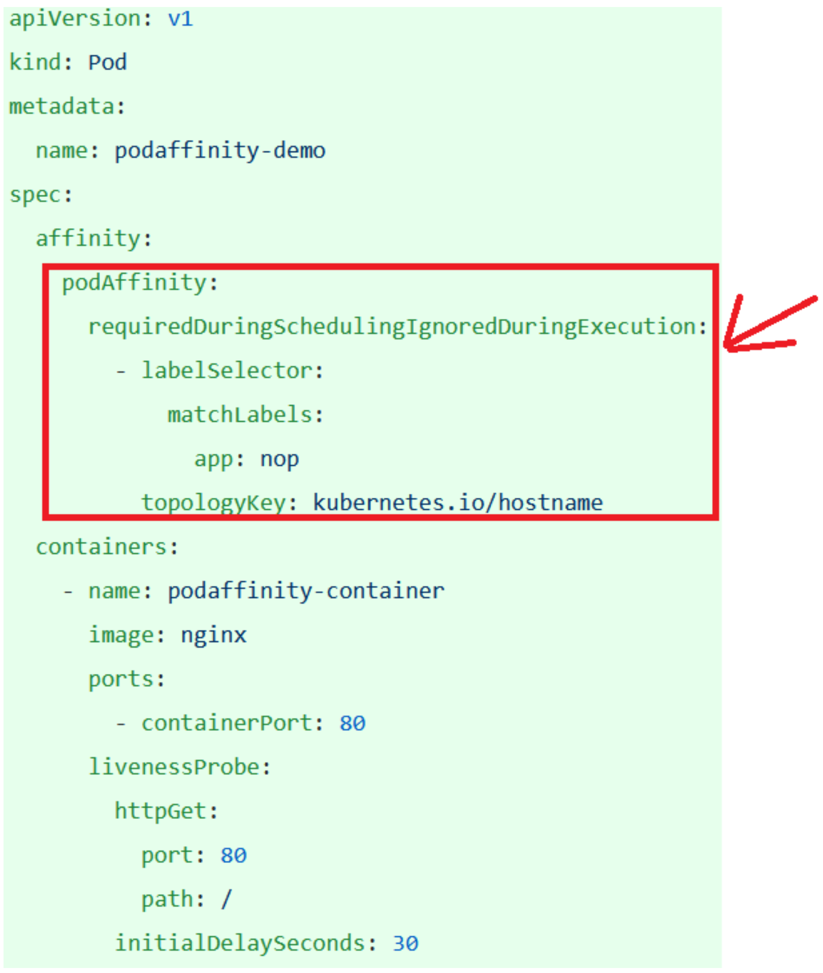
Pod Affinity and Antiaffinity (Automated Placement)

- Pod Affinity defines the expression where we can schedule the Pod depending on the other Pod i.e. try to run Pod B on the same node where Pod A is running (So that they can use localhost for faster communication.)
- Pod Antiaffinity is try to run Pod B on the Node where Pod A is not running
- Refer below for the Pod Affinity Example



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: podaffinity-demo
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: nop
          topologyKey: kubernetes.io/hostname
  containers:
    - name: podaffinity-container
      image: nginx
      ports:
        - containerPort: 80
      livenessProbe:
        httpGet:
          port: 80
          path: /
        initialDelaySeconds: 30
```

Here, We are specifying create a new pod where pod with labels nop is already running.

Taints and Tolerations

- Node affinity is a property of Pod that allows them to choose nodes while taints and tolerations are opposite. They allow Nodes to control which Pods should or should not be scheduled.
- A taint is characteristic of node, when it is present, it prevents Pods from scheduling onto to the node unless Pod has toleration for the taint.(Best Example of it is master node where because of taint no node is scheduled is master node.)
- By default master has a taint with effect No Schedule

Tainted node

```
apiVersion: v1
kind: Node
metadata:
  name: master
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
```

- Pod tolerating node taints

```
apiVersion: v1
kind: Pod
metadata:
  name: taint-tolerate-demo
spec:
  containers:
    - image: nginx
      name: tt-container
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
```

There are hard taints prevent schedulint on a node (effect=NoSchedule), soft taints try to avoid scheduling on a node (effect=PreferNoSchedule) and taints that can evict running Pods from a Node (effect=NoExecute)

Behaviroal Patterns

- The following are popular behaviroal patterns
  - Batch Job
  - Periodic Job
  - Daemon Service
  - Singleton Service
  - Stateful Service
  - Service Discovery
  - Self Awarness

Batch Job

a batch job is a scheduled program that is assigned to run on a computer without further user interaction

- This Pattern is suited for managing isolate atomic units of work. It is based on Job abstraction which runs short-lived Pods reliably until completed on a distribute environment

Problem:

- The different ways of creating Pods with Varying characteristics
  - Bare Pod
  - ReplicaSets
  - DaemonSet
- A common characteristic aspect of these Pods is the fact that they represent long-running processes that are not meant to stop after some time. However in some cases there is need to perform predefined finite unit of work reliably and then shutdown the container.

- Solution:
  - K8s has a Job resource.
  - Job is similar to Replica Set as it creates one or more Pods and ensures they run successfully. However, the difference is that once the expected number of Pods terminate successfully, the Job is considered complete and no additional Pods will be created.

```yaml
---
apiVersion: batch/v1
kind: Job
metadata:
  name: jobdemo
spec:
  completions: 6
  parallelism: 2
  template:
    metadata:
      name: jobdemo-pod
    spec:
      restartPolicy: OnFailure
      containers:
        - image: alpine
          name: jobdemo-c
          command: ["sleep", "1h"]
```

This Job will take 3 hrs and few minutes to complete.

Here we are going to have 6 completion where at a time 2 jobs would be running.

Example: Cluster-restart

Periodic Job

- This pattern extends the Batch Job Pattern by adding a time dimension and allowing unit of work to be triggered by an event.
- Problem: We need to Perform some automated or business tasks which take finite time to run on a schedule

Solution:

```yaml
---
apiVersion: batch/v1
kind: CronJob
metadata:
  name: cronjobdemo
spec:
  schedule: "0 0 * * 0,6"
  jobTemplate:
    spec:
      template:
        metadata:
          name: cronjobdemo-pod
        spec:
          restartPolicy: OnFailure
          containers:
            - image: alpine
              name: cronjobdemo-c
              command: ["sleep", "1h"]
```