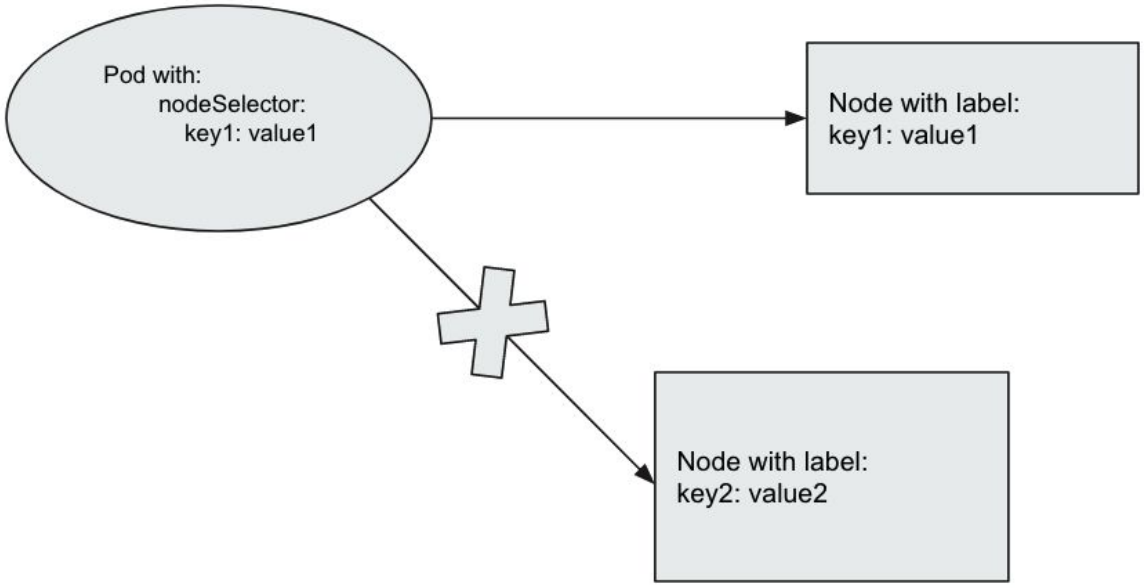
Resources Management and Pod Assignment

Node Selector

- nodeSelector is the simplest recommended form of node selection constraint. nodeSelector is a field of PodSpec.
- It specifies a map of key-value pairs.
- For the pod to be eligible to run on a node, the node must have each of the indicated key-value pairs as labels (it can have additional labels as well). The most common usage is one key-value pair.



Node Affinity and Anti Affinity

- Node affinity, is a property of Pods that attracts them to a set of nodes (either as a preference or a hard requirement).
- nodeSelector provides a very simple way to constrain pods to nodes with particular labels.
- The affinity/anti-affinity feature, greatly expands the types of constraints you can express.
- The key enhancements are The affinity/anti-affinity language is more expressive.

- The language offers more matching rules besides exact matches created with a logical AND operation; you can indicate that the rule is "soft"/"preference" rather than a hard requirement, so if the scheduler can't satisfy it, the pod will still be scheduled
- you can constrain against labels on other pods running on the node (or other topological domain), rather than against labels on the node itself, which allows rules about which pods can and cannot be co-located The affinity feature consists of two types of affinity, "node affinity" and "inter-pod affinity/anti-affinity".

Node Affinity Types

There are currently two types of node affinity, called

1. requiredDuringSchedulingIgnoredDuringExecution
2. preferredDuringSchedulingIgnoredDuringExecution.

You can think of them as "hard" and "soft" respectively, in the sense that the former specifies rules that must be met for a pod to be scheduled onto a node. while the latter specifies preferences that the scheduler will try to enforce but will not guarantee.
The "IgnoredDuringExecution" part of the names means that, similar to how nodeSelector works, if labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod will still continue to run on the node.

Node Taints

- Taints allow a node to repel a set of pods.
- You add a taint to a node using kubectl taint. For example:

```
kubectl taint nodes node1 key=value:NoSchedule
```

- That means that a pod with label key=value will be rejected to scheduling by the node.

Resources - Limits and Requests

- If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its request for that resource specifies.
- However, a container is not allowed to use more than its resource limit.
- For example, if you set a memory request of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.
- If you set a memory limit of 4GiB for that Container, the kubelet (and container runtime) enforce the limit. The runtime prevents the container from using more than the configured resource limit.

- For example: when a process in the container tries to consume more than the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error.

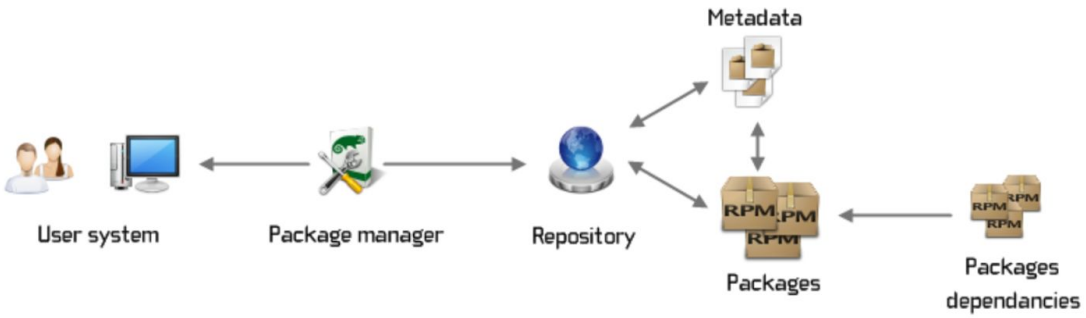
```yaml
containers:
- name: db
  image: mysql
  env:
  - name: MYSQL_ROOT_PASSWORD
    value: "password"
  resources:
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
```

Example

follow steps at: k8s-experts/assigning-pods-to-nodes/commands.txt

Helm, the Package Manager for Kubernetes

- A package management system is a collection of tools that provides a consistent method of installing, upgrading and removing software on your system.
- Most every programming language and operating system has its own package manager to help with the installation and maintenance of software. Many of the package managers you may already be familiar with, such as Debian's apt, or Python's pip.
- Software is distributed through Packages that are linked to metadata which contain additional information such as a description of the software purpose and a list of dependencies necessary for the software to run properly.
- Packages are archives of files that include all the files making up a piece of software (such as an application itself, shared libraries, development packages containing files needed to build software against a library, ...) and, eventually, instructions on the way to make them work.



Helm Package Manager

- Deploying applications to Kubernetes – the powerful and popular container-orchestration system – can be complex.
- Setting up a single application can involve creating multiple interdependent Kubernetes resources – such as pods, services, deployments, and replicasets – each requiring you to write a detailed YAML manifest file.
- Helm is a package manager for Kubernetes that allows developers and operators to more easily package, configure, and deploy applications and services onto Kubernetes clusters.
- Helm is now an official Kubernetes project and is part of the **Cloud Native Computing Foundation**, a non-profit that supports open source projects in and around the Kubernetes ecosystem.

Helm can:

- Install software.
- Automatically install software dependencies.
- Upgrade software.
- Configure software deployments.
- Fetch software packages from repositories.

Helm provides this functionality through the following components:

- A command line tool, helm, which provides the user interface to all Helm functionality.
- A companion server component, tiller, that runs on your Kubernetes cluster, listens for commands from helm, and handles the configuration and deployment of software releases on the cluster.
- The Helm packaging format, called charts.
- An official curated charts repository with prepackaged charts for popular open-source software projects.

Helm Charts

Helm packages are called charts, and they consist of a few YAML configuration files and some templates that are rendered into Kubernetes manifest files. Here is the basic directory structure of a chart:

```
package-name/
  charts/
  templates/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
```

- **charts/**: Manually managed chart dependencies can be placed in this directory, though it is typically better to use requirements.yaml to dynamically link dependencies.
- **templates/**: This directory contains template files that are combined with configuration values (from values.yaml and the command line) and rendered into Kubernetes manifests. The templates use the Go programming language's template format.
- **Chart.yaml**: A YAML file with metadata about the chart, such as chart name and version, maintainer information, a relevant website, and search keywords.
- **requirements.yaml**: A YAML file that lists the chart's dependencies.
- **values.yaml**: A YAML file of default configuration values for the chart.

- The helm command can install a chart from a local directory, or from a **.tar.gz** packaged version of this directory structure.
- These packaged charts can also be automatically downloaded and installed from chart repositories or repos.
- A Helm chart repo is a simple HTTP site that serves an index.yaml file and .tar.gz packaged charts. The helm command has subcommands available to help package charts and create the required index.yaml file. These files can be served by any web server, object storage service, or a static site host such as GitHub Pages.

Helm Predefined Values

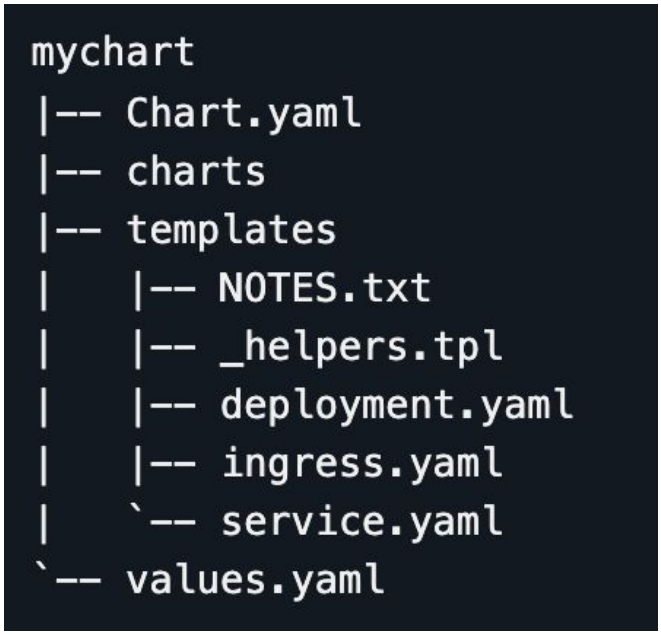
- Values that are supplied via a values.yaml file (or via the --set flag) are accessible from the .Values object in a template.
- A values file is formatted in YAML.
- A chart may include a default values.yaml file.
- The Helm install command allows a user to override values by supplying additional YAML values as we will see in the upcoming demo.

Helm Installation and Demo

1. Follow instructions under k8s-demo/helm/helm-init-commands.txt
    a. curl -L https://git.io/get_helm.sh | bash
    b. helm init # setup helm with our cluster
    c. helm repo update # sync all helm charts info
    d. helm repo list
    e. helm list
2. Follow instructions under k8s-demo/helm/deploy-helm-chart.txt
3. Now let's review the values.yaml
4. But how all of this changes are set to the application?

Creating Helm Chart

The best way to get started with a new chart is to use the helm create command to scaffold out an example we can build on. Use this command to create a new chart.
helm create mychart



Templates

- The most important piece of the puzzle is the templates/ directory.
- This is where Helm finds the YAML definitions for your Services, Deployments and other Kubernetes objects.
- If you already have definitions for your application, all you need to do is replace the generated YAML files for your own.
- What you end up with is a working chart that can be deployed using the helm install command.

This is a basic Service definition using templating. When deploying the chart, Helm will generate a definition that will look a lot more like a valid Service. We can do a dry-run of a helm install and enable debug to inspect the generated definitions:

```
helm install --dry-run --debug ./mychart
...
# Source: mychart/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
name: pouring-puma-mychart
labels:
    chart: "mychart-0.1.0"
spec:
type: ClusterIP
ports:
- port: 80
    targetPort: 80
    protocol: TCP
    name: nginx
selector:
    app: pouring-puma-mychart
...
```

Values.yaml

- The template makes use of the Helm-specific objects .Chart and .Values.
- .Chart provides metadata about the chart to your definitions such as the name, or version.
- .Values object is a key element of Helm charts, used to expose configuration that can be set at the time of deployment.
- The defaults for this object are defined in the values.yaml file.

Developing Helm Chart

- follow the steps at helm/dev-and-deploy-chart.txt
    a. helm create mychart
    b. helm install ./mychart --dry-run --debug --name mychart
    c. helm install ./mychart --name mychart
    d. helm package ./mychart
    e. helm upgrade --install mychart mychart-0.1.0.tgz --set replicaCount=3
    f. kubectl get pods

Helm Chart Development Guide

https://medium.com/swlh/helm-chart-development-guide-bbc525d3b448

Helm Chart Repository

- A chart repository is an HTTP server that houses an index.yaml file and optionally some packaged charts.
- When you're ready to share your charts, the preferred way to do so is by uploading them to a chart repository.
- Because a chart repository can be any HTTP server that can serve YAML and tar files and can answer GET requests, you have a variaty of options when it comes down to hosting your own chart repository.
- For example, you can use a Google Cloud Storage (GCS) bucket, Amazon S3 bucket, Github Pages, or even create your own web server.
- ChartMuseum - ChartMuseum is an open-source, easy to deploy, Helm Chart Repository server.
- To deploy a chart museum repository on our minikube cluster go to helm folder and run deploy-repository.txt commands

Kubernetes Infrastructure Deployments

Kubernetes Automation

When considering which kubernetes installer to choose, these are the key points we need to look at:

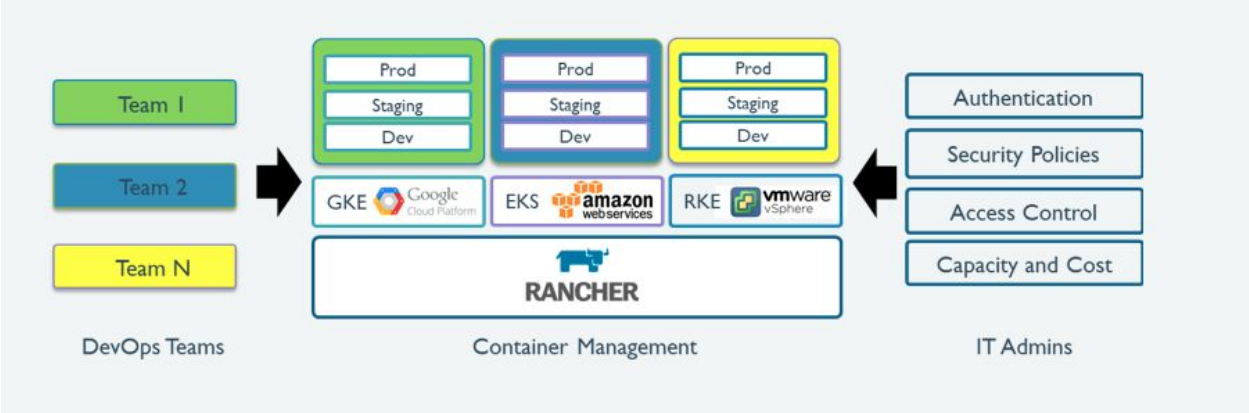
- Usable for On-Premise systems
- Possibility to automate installation
- Able to deliver high available setups
- Possibility to configure cluster as far as needed
- Open Source and well maintained
- Easy to use and straight forward
- Well documented
- Often used

There are several tools out there to set up a Kubernetes Cluster.

- kubeadm: can be used to deploy a single master or an HA cluster
- Kubespray: based on the Ansible playbook and uses kubeadm behind the scenes to deploy single or multi masters clusters
- eksctl: dedicated to deploying a cluster on AWS infrastructure
- Rancher: provides a great web UI to manage several clusters from one location

Rancher

Rancher is open source software that combines everything an organization needs to adopt and run containers in production.
Built on Kubernetes, Rancher makes it easy for DevOps teams to test, deploy and manage their applications.



- Authorization and Role-Based Access Control
  - User management
  - Authorization
- Working with Kubernetes
  - Provisioning kubernetes clusters
  - Catalog management
  - Managing projects
  - Pipelines
- Cluster Manageability
  - Logging
  - Monitoring
  - Backup and Recovery

Follow steps described at: k8s-experts/on-premise/rancher/commands.txt to create a fully scaled kubernetes cluster with rancher.