Manifest and Templating

What is a Manifest

It's basically a Kubernetes "API object description". A config file can include one or more of these. (i.e. Deployment, ConfigMap, Secret, DaemonSet, etc).

Every manifest will have at least these four field:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
```

The apiVersion: field specifies the API group you want to use to create the resource and the version of the API to use.

```
$ kubectl api-versions |more
```

```
admissionregistration.k8s.io/v1beta1
apiextensions.k8s.io/v1beta1
apiregistration.k8s.io/v1
apiregistration.k8s.io/v1beta1
...
```

The second line, "kind:", lists the type of resource you want to create. Deployments, ReplicaSets, CronJobs, StatefulSet, etc. are examples of resources you can create.

```
$ kubectl api-resources |more
```

```
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
daemonsets                        ds           apps                           true         DaemonSet
deployments                       deploy       apps                           true         Deployment
replicasets                       rs           apps                           true         ReplicaSet
statefulsets                      sts          apps                           true         StatefulSet
...
```

The metadata: section is used to uniquely identify the resource inside a Kubernetes cluster. This is were you name the resource, assign tags, annotations, specify a namespace, etc.

```
$ kubectl explain deployment.metadata | more
```

```
KIND:     Deployment
VERSION:  extensions/v1beta1

RESOURCE: metadata <Object>

DESCRIPTION:
     Standard object metadata.

     ObjectMeta is metadata that all persisted resources must have, which
     includes all objects users must create.

FIELDS:
   annotations  <map[string]string>
     Annotations is an unstructured key value map stored with a resource that
     may be set by external tools to store and retrieve arbitrary metadata. They
     are not queryable and should be preserved when modifying objects. More
     info: http://kubernetes.io/docs/user-guide/annotations
...
```

The spec section describes how to create and manage a resource. You will define the container image to use, the number of replicas in a ReplicaSet, the selector criteria, liveness and readiness probe definitions.

```
$ kubectl explain deployment.spec | more
```

```
KIND:     Deployment
VERSION:  extensions/v1beta1

RESOURCE: spec <Object>

DESCRIPTION:
     Specification of the desired behavior of the Deployment.

     DeploymentSpec is the specification of the desired behavior of the
     Deployment.

FIELDS:
   minReadySeconds      <integer>
     Minimum number of seconds for which a newly created pod should be ready
     without any of its container crashing, for it to be considered available.
     Defaults to 0 (pod will be considered available as soon as it is ready)
...
```

What is Templating?

You've deployed your application and exposed it via a service. Now what? Kubernetes provides a number of tools to help you manage your application deployment, including scaling and updating.

If you are working with Kubernetes environment then you probably make use of several existing templating tools, some of them being a part of package managers such as Helm or Ksonnet or just templating languages (Jinja2, Go, Ansible etc.)

Helm

Helm today is de-facto the standard for Kubernetes applications packaging. The main advantage of Helm is large community and a big number of public repositories with charts. And recently Helm developers have announced a Helm Hub. So Helm today is like Docker — it's not the only one but it has community and support.

To conclude, Helm advantages:

- Large community and a number of public charts
- (Relatively) human-friendly syntax.

Drawbacks:

- Working with strings and not objects
- Limited number of operators and functions you can use

```
spec:
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: my-awesome-container
            resources:
{{ toYaml .Values.resources | indent 14 }}
```

Ksonnet

Ksonnet is a tool based on JSON templating language Ksonnet. Another cool feature about Ksonnet is that it has Kubernetes-API-compatible Ksonnet libraries that you can import into your template and work with Kubernetes objects like in any OOP language.

In total, advantages of Ksonnet:

- Working with objects
- Kubernetes-API-compatible libraries
- Helm chart import support

Drawbacks:

- Smaller community and smaller number of Ksonnet-native packages
- Lack of some functionality you can use in Helm
- New syntax => increased learning time => increased bus-factor
- Syntax can sometimes get ugly and less human-readable (especially when making workarounds for lacking features)

```
{
  global: {},
  components: {
    "deployment-nginx-deployment-dkecx"+: {
      spec+: {
        replicas: 10,
        template+: {
          spec+: {
            containers+: [
              {
                name: "nginx",
                image: "nginx:latest",
                ports: [
                  {
                    containerPort: 80,
                  },
                ],
              },
            ],
          },
        },
      },
    },
  },
}
```

Kustomize

Kustomize isn't a new tool, it is under construction since 2017 and has been introduced as a native Kubernetes sub-command in the version 1.14. Kustomize is like Kubernetes, it is totally declarative. You say what you want and the system provides it to you. You don't have to follow the imperative way and describe how you want it to build the thing.

Secondly, it works like Docker. You have many layers and each of those is modifying the previous ones. Thanks to that, you can constantly write things above others without adding complexity inside your configuration. The result of the build will be the addition of the base and the different layers you applied over it.

Lastly, like Git, you can use a remote base as the start of your work and add some customization on it.