Advanced Kubernetes

WRITTEN BY CHRIS GAUN PRODUCT MANAGER FOR MESOSPHERE KUBERNETES ENGINE
AND JOE THOMPSON SOLUTIONS ARCHITECT FOR MESOSPHERE KUBERNETES ENGINE

CONTENTS

About Kubernetes

Kubernetes is a distributed cluster technology that manages container-based systems in a declarative manner using an API. Kubernetes's core open-source source code is governed by the Cloud Native Computing Foundation (CNCF) and counts all the largest cloud providers and software vendors as its contributors. There are currently many learning resources to get started with the fundamentals of Kubernetes, but there is less information on how to manage Kubernetes infrastructure on an ongoing basis. This Refcard aims to deliver quickly accessible information for operators using any Kubernetes product.

For an organization to deliver and manage Kubernetes clusters to every line of business and developer groups, ops needs to architect and manage both the core Kubernetes container orchestration and the necessary auxiliary solutions — e.g. monitoring, logging, and the CI/CD pipeline.

Gartner predicts that by 2020, more than 50% of global organizations will be running containerized applications in production, up from less than 20% today. This research provides an overview and actionable advice for organizations implementing modern containerized apps, data services, and machine learning.

According to 451 Research, the application container market size was $762 million last year and will grow by over 3.5x to $2.7 billion by 2020. Organizations can prepare for the rapid growth in containerized applications, real-time data, and machine learning by taking steps to properly architect the network.

Kubernetes differs from the orchestration offered by configuration management solutions in that it provides a declarative API that collects, stores, and processes events in an eventually consistent manner.

A few traits of Kubernetes include:

- **Abstraction**: Kubernetes abstracts the application orchestration from the infrastructure resource and as-a-service automation. This allows organizations to focus on the APIs of Kubernetes to manage an application at scale in a highly available manner instead of the underlying infrastructure resources.
- **Declarative**: Kubernetes's control plane decides how the hosted application is deployed and scaled on the underlying fabric. A user simply defines the logical configuration of the Kubernetes object, and the control plane takes care of the implementation.

KUBERNETES DOESN'T HAVE TO BE ROCKET SCIENCE.

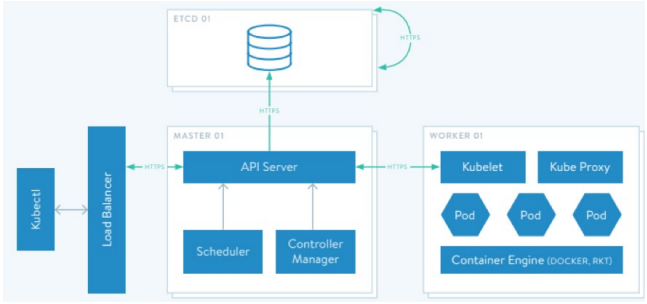
The Kubernetes Cheatsheet is here to help

Download Now →

MESOSPHERE

- **Immutable**: Different versions of services running on Kubernetes are completely new and not swapped out. Objects in Kubernetes, say different versions of a pod, are changed by creating new objects.

Basics



Simplified Kubernetes architecture and components

CONTROL PLANE, NODES, AND PERSISTENT STORAGE

Kubernetes's basic architecture requires a number of components.

API Server: The main way to communicate with Kubernetes clusters is through the Kubernetes API Server. A user (and other Kubernetes components) sends declarative events to the Kubernetes API Server through HTTPS (port 443). The API Server then processes events and updates the Kubernetes persistent data store — e.g. etcd. The API Server also performs authentication and authorization.

For more information, see the Kubernetes documentation here.

etcd Cluster: As Kubernetes collects declarative events, it needs to store them. Kubernetes uses etcd for this persistent storage, providing a single source of truth of all Kubernetes objects. etcd requires an odd number of nodes. A loss of etcd storage results in a loss of the cluster's state, so etcd should be backed up for disaster recovery. For more information, see the Kubernetes documentation here.

Controller Manager: Kubernetes uses a control plane to perform non-terminating control loops to observe the state of Kubernetes objects and reconcile it with the desired state. For more information, see the Kubernetes documentation here.

Kubelet (Agent): Kubernetes clusters have a worker node called a Kubelet that has several functions. For example, Kubelets create Pods from the Podspec. For more information, see the Kubernetes documentation here.

Scheduler: Kubernetes relies on a sophisticated algorithm to schedule Kubernetes objects. The scheduling takes into account filters such as resources, topology, and volume, then uses prioritization setting, such as affinity rules to provision pods on particular Kubernetes worker nodes. For more information, see the Kubernetes documentation here.

Kube-Proxy: Each Kubernetes cluster worker nodes have a network proxy for connectivity. For more information, see the Kubernetes documentation here.

OTHER STANDARD COMPONENTS OF KUBERNETES

In order to run the most basic Kubernetes cluster, a number of additional components and add-ons are required.

Kube-DNS: All services and pods (when enabled) on Kubernetes are assigned a domain name that is resolved by the DNS. This DNS is handled by Kube-DNS, which itself is a pod and a service on Kubernetes. Kube-DNS resolves DNS of all services in the clusters.

Kubectl: The official command line for Kubernetes is called kubectl (the pronunciation is up for debate). All industry standard Kubernetes commands start with `kubectl`.

Metrics Server and Metrics API: Kubernetes has two resources for giving usage metrics to users and tools. First, Kubernetes can include the metrics server, which is the centralized aggregation point for Kubernetes metrics. The second is the Kubernetes metrics API, which can provide an API to access these aggregated metrics.

Web UI (Dashboard): Kubernetes has an official GUI called Dashboard. That GUI is distinct from the single-vendor GUIs that have been designed for specific Kubernetes derivative products.

Note that the Kubernetes Dashboard release version does not necessarily match the Kubernetes release version.

KUBERNETES CONSTRUCTS

Kubernetes has a number of constructs for defining and managing objects on the cluster.

Namespaces: Kubernetes includes a means to segment a single physical cluster into separate logical clusters using namespacing.

Pods: Whatever the runtime, Kubernetes fundamentally manages a logical grouping of one or more containers called a pod.

StatefulSets: Kubernetes controllers for managing workloads that require proper management of state.

ReplicaSet: One of the control loops available in Kubernetes that ensures that the desired number of pods are running

Roles: Kubernetes has several access control schemes, and users should always default to role-based access controls to maximize security.

Ingresses and Load Balancing: In order to expose a service outside a cluster, a user should set up an ingress for layer 7 or define the configuration of a layer 4 load balancer using `"type=loadbalancer"` in the service definition.

Deployments: The declarative controller in Kubernetes that manages replicasets of pods.

Services: Defined by a label, a Kubernetes service is a logical layer that provides IP/DNS/etc. persistence to dynamic pods.

DaemonSet: A Kubernetes construct that enables users to run a pod on every node in the cluster.

Jobs and Cronjobs: Kubernetes includes the logic to run jobs, processes that run to completion, and cronjobs — processes that run at specific intervals and run to completion.

Extension Points: Kubernetes has a number of points to extend its core functionality.

Custom Resource Definition (CRD): CRD allows users to extend Kubernetes with custom APIs for different objects beyond the standard ones supported by Kubernetes.

Container Runtime Interface (CRI): CRI is a plugin API that enables Kubernetes to support other container runtimes beyond Docker and Containerd.

Container Network Interface (CNI): CNI gives users a choice of network overlay that can be used with Kubernetes to add SDN features.

Container Storage Interface (CSI): CSI empowers users to support different storage systems through a driver model.

Kubectl

Below are some useful commands for IT professionals getting started with Kubernetes. A full list of Kubectl commands can be found at the reference documentation.

MAKING LIFE EASIER

Finding Kubernetes command short name:

```
kubectl describe
```

You can find out more about using Kubectl Aliases here and context switching among Kubernetes clusters here.

Setting your Bash prompt to mirror Kubernetes context:

```
prompt_set() {
if [ "$KUBECONFIG" != "" ]; then
  PROMPT_KUBECONTEXT="k8s:$(kubectl config current-
context 2>/dev/null)\n"
```

CODE CONTINUED ON FOLLOWING COLUMN

```
fi
PS1="${PROMPT_KUBECONTEXT}[\u@\h \W]\$ "
}
K8s:admin@local
[kensey@sleeper-service ~]$
```

COMMANDS

Print the version of the API Server:

```
$ kubectl API version
```

IP addresses of master and services:

```
$ kubectl cluster-info
```

List all the namespaces used in Kubernetes:

```
$ kubectl cluster-info dump --namespaces
```

Mark a node as unschedulable (used for maintenance of cluster):

```
$ kubectl cordon NODE
```

Mark a node as scheduled (used after maintenance):

```
$ kubectl uncordon NODE
```

Removes pods from a node via graceful termination for maintenance:

```
$ kubectl drain NODE
```

Find the names of the objects that will be removed:

```
$ kubectl drain NODE --dry-run=true
```

Removes pods even if they are not managed by controller:

```
$ kubectl drain NODE --force=true
```

Taint a node so they can only run dedicated workloads or certain pods that need specialized hardware:

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

Start an instance of Nginx:

```
$ kubectl run nginx --image=nginx --port=8080
```

Print information on Kubernetes resources, including `all`, `certificatesigningrequests` (AKA `csr`), `clusterrolebindings`, `clusterroles`, etc:

```
$ kubectl get RESOURCE
```

Print the documentation of resources:

```
$ kubectl explain RESOURCE**
```

Scale a ReplicaSet (rs) named `foo`; can also scale a replication controller or StatefulSet:

```
$ kubectl scale \--replicas=COUNT rs/foo
```

Perform a rolling update:

```
$ kubectl rolling-update frontend-v1 -f frontend-v2.
json
```

Update the labels of resources:

```
$ kubectl label pods foo GPU=true
```

Delete `foo` pods:

```
$ kubectl delete pod foo
```

Delete `foo` services:

```
$ kubectl delete svc foo
```

Create a clusterIP for a service named `foo`:

```
$ kubectl create service clusterip foo \--
tcp=5678:8080
```

Autoscale pod `foo` with a minimum of 2 and maximum of 10 replicas when CPU utilization is equal to or greater than 70%:

```
$ kubectl autoscale deployment foo \--min=2 \--
max=10 \--cpu-percent=70
```

Managing Nodes

Sometimes, it is necessary to perform maintenance on underlying nodes. In those cases, it is important to use eviction and make sure the application owners have set a pod disruption budget. To properly evict a node, use:

1. **Cordon node**: `kubectl cordon $NODENAME`
2. **Drain node**: `kubectl drain`
   - Respects `PodDisruptionBudgets`
3. **Uncord node**: `kubectl uncordon $NODENAME`

Test Plan

According to the CNCF, most Kubernetes clusters are still provisioned and managed software distributions (instead of public cloud provider options). For these clusters, once a new cluster is up and running, it is imperative to devise a test plan and design a run book to make sure it is operational. For new public cloud providers or new versions of Kubernetes on a provider, it is also important to run a testing plan.

Below is testing plan used as part of Mesosphere's testing of Kubernetes. After each step, validate that the cluster is working properly:

Test clusters:

1. Provision a highly available Kubernetes cluster with 100 Kubernetes nodes.
2. Scale that cluster down to 30 nodes after it has finished provisioning.
3. Provision 3 additional highly available Kubernetes clusters with 5 nodes.
4. Scale all 3 clusters (in parallel) to 30 nodes simultaneously.
5. Provision 16 more highly available Kubernetes clusters with 30 nodes.
6. Kill 5 Kubernetes nodes on a single cluster simultaneously.
7. Kill 3 control-plane nodes on a single cluster (fewer if the nodes will not automatically reprovision).
8. Kill the `etcd` leader.

Test workloads:

1. Provision storage (potentially using CSI and a specific driver).
   - Test different provider specific storage features.
2. Run the e2e cluster loader with higher end of pods per node (35-50).
3. Run Kubernetes conformance testing to stress the cluster.
4. Provision Helm to the Kubernetes cluster.
5. Test specific Kubernetes workloads (using Helm charts).
   - Deploy Nginx
   - Deploy Redis
   - Deploy Postgres
   - Deploy RabbitMQ
6. Deploy services with `type=loadbalancer` to test load balancer automation.
   - Test a different provider's specific load balancer features.
7. Expose a service using Ingress.

Security and Troubleshooting

TROUBLESHOOTING WITH KUBECTL

The Kubernetes command line kubectl provides many of the resources needed in order to debug clusters.

CHECK PERMISSIONS

TROUBLESHOOTING	COMMAND	EXAMPLE
Check your permissions	`kubectl auth can-i`	`kubectl auth can-i create deployments--namespace dev`
Check permissions of other uses	`kubectl auth can-i [Options]`	`kubectl auth can-i create deployments--namespace dev chris`

PENDING CHECK RESOURCES:

TROUBLESHOOTING	COMMAND
General issue checking	`kubectl describe pod <name of pending pod>`
Check to see if pod is using too much resources	`kubectl top pod`
Check node resources	`kubectl top node`
Get node resources	`kubectl get nodes -o yaml | grep`
Get all pod resources	`kubectl top pod --all-namespaces --containers=true`

You can also remove pods from the environment.

WAITING INCORRECT IMAGE INFORMATION

TROUBLESHOOTING	COMMAND
Check YAML URL	`spec: containers: – name: example image: url:port/image:v`
Pull an image onto desktop to determine whether registry and image information is correct	`docker pull <image>`

Also check that the secret information is correct.

CRASH LOOPING DEPLOYMENT

- Use ctrl + C to exit the crash loops
- Troubleshoot
- Roll back deployment

TROUBLESHOOTING	COMMAND
General issue checking	`kubectl describe deployments`
Rolling back deployment	`kubectl rollout undo [Deployment Name]`
Pausing a deployment	`kubectl rollout pause [Deployment Name]`

General Debugging

Watch the `stdout` of a container:

```
kubectl attach bash-rand-77d55b86c7-bntxs
```

Copy files into and out of containers (note: requires a tar binary in the container):

```
$ kubectl cp default/bash-rand-77d55b86c7-bntxs:/root/rand .
```

CODE CONTINUED ON FOLLOWING COLUMN

```
tar: Removing leading `/' from member names

$ cat rand
567bea045d8b80cd6d007ced02849ac4


Kubectl -v – Increase the verbosity of kubectl output (99 is the de facto “get everything”):

$ kubectl -v 99 get nodes
I1211 11:10:28.611959   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.612482   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.614383   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.617867   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.629567   24842 round_trippers.go:405] GET https://192.168.122.138:6443/api/v1/nodes?limit=500 200 OK in 11 milliseconds
I1211 11:10:28.630279   24842 get.go:558] no kind is registered for the type v1beta1.Table in scheme "k8s.io/kubernetes/pkg/api/legacyscheme/scheme.go:29"
NAME             STATUS   ROLES    AGE   VERSION

Kubectl explain – Getting help on the fly:
$ kubectl explain crd
KIND:     CustomResourceDefinition
VERSION:  apiextensions.k8s.io/v1beta1

DESCRIPTION:
     CustomResourceDefinition represents a resource that should be exposed on
     the API server. Its name MUST be in the format
<.spec.name>.<.spec.group>.

FIELDS:
[...]

Output control using `kubectl [command] -o [format]` – render [command] output as [format]:
$ kubectl get deploy bash-rand -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2018-12-11T05:20:32Z
  generation: 1
  labels:
    run: bash-rand
[...]
```

Troubleshooting With Curl

Adding headers to requests are often used for:

- Setting accepted content types for replies
- Setting content type of posted content
- Injecting bearer auth tokens

Building and submitting requests:

```
GET: request is contained in the URL itself (default
method) — used to read/list/watch resources
POST: submit a data blob to create resources
PATCH: submit a data blob to merge-update resources
PUT: submit a data blob to replace a resource
DELETE: submit options to control deletion of a
resource
        $ curl --cert client.cert --key client.key
--cacert cluster-ca.cert \
         https://192.168.100.10:6443/api/v1/
namespaces/default/pods
          {
            "kind": "PodList",
             "apiVersion": "v1",
             "metadata": {
              "selfLink": "/api/v1/namespaces/
default/pods",
            "resourceVersion": "126540"
             },
          "items": [
```

Feeding Results to Other Things

Basic method (`bash while read loop`):

```
$ while read -r serverevent; do echo "$serverevent"
| jq '.["operation"] = .type | .["node"] = .object.
metadata.name | { "operation": .operation, "node":
.node}' ;
done < <(curl -sN --cert client.cert --key client.
key --cacert cluster-ca.cert
https://192.168.100.10:6443/api/v1/nodes?watch=true)
{
  "operation": "ADDED",
  "node": "192.168.100.10"
}
[...]
```

Something completely different (`bash coproc`):

```
#!/bin/bash

coproc curl -sN --cacert cluster-ca.cert --cert ./
client.cert --key ./client.key \
https://192.168.100.10:6443/api/v1/nodes?watch=true
```

CODE CONTINUED ON FOLLOWING COLUMN

```
exec 5<&${COPROC[0]}

while read -ru 5 serverevent; do
  if [[ $(echo $serverevent | jq -r '.type') ==
"ADDED" ]]; then
    echo "Added node $(echo $serverevent | jq -r
'.object.metadata.name') in namespace \
$(echo $serverevent | jq '.object.metadata.
namespace')"
  fi
done

trap 'kill -TERM $COPROC_PID' TERM INT
```

Troubleshooting With jq

Basic usage:

```
[some JSON content] | jq [flags] [filter
expression]
```

The dot filter (prettyprints input unchanged):

```
$ echo '{ "key": "value", "key2": { "subkey":
"value", "subkey2": "value2" }, "key3": 12, "key4":
[ "one", "two" ] }' | \
jq .
{
  "key": "value",
  "key2": {
    "subkey": "value",
    "subkey2": "value2"
  },
  "key3": 12,
  "key4": [
    "one",
    "two"
  ]
}
```

Using the hierarchy to filter the input:

```
$ echo '{ "key": "value", "key2": { "subkey":
"value", "subkey2": "value2" }, "key3": 12, "key4":
[ "one", "two" ] }' | \
jq .key2
{
  "subkey": "value",
  "subkey2": "value2"
}
 $ echo '{ "key": "value", "key2": { "subkey":
"value", "subkey2": "value2" }, "key3": 12, "key4":
[ "one", "two" ] }' | \
jq .key4[1]
"two"
```

More advanced functions:

- `select`: Cherry-pick a piece of the input by criteria.
- `contains`: Match a value that contains an element.

```
$ echo '[ { "key": "value", "key2": { "subkey":
"value", "subkey2": "value2" }, "key3": 12, "key4":
[ "one", "two" ] }, { "key": "value", "key2": {
"subkey": "value3", "subkey2": "value4" }, "key3":
12, "key4": [ "three", "four" ] } ]' | \
jq '.[] | select( .key2.subkey | contains ( "value3"
) )'
{
  "key": "value",
  "key2": {
    "subkey": "value3",
    "subkey2": "value4"
  },
  "key3": 12,
  "key4": [
    "three",
    "four"
  ]
}
```

Scripting beyond one-liners (jq allows creating a script file for readability — also great for easier source control):

```
$ cat script.jq
.[]
| select( .key2.subkey
| contains ( "value3" ) )

$ echo '[ { "key": "value", "key2": { "subkey":
"value", "subkey2": "value2" }, "key3": 12, "key4":
[ "one", "two" ] }, { "key": "value", "key2": {
"subkey": "value3", "subkey2": "value4" }, "key3":
12, "key4": [ "three", "four" ] } ]' | \
jq -f script.jq
{
  "key": "value",
  "key2": {
    "subkey": "value3",
    "subkey2": "value4"
  },
  "key3": 12,
  "key4": [
    "three",
    "four"
  ]
}
```

Security

Security is fundamental for Kubernetes day two operations. Users should make sure that some basic security for the clusters is being maintained. Here is a security checklist for Kubernetes administrators.

Newest Kubernetes patch release, i.e. 1.x.y (make sure it has all CVE fixes): A new version of Kubernetes is released every three months and patches come out at regular intervals for the latest three versions of Kubernetes. Each patch release has bug fixes for Kubernetes. Because of the lack of widespread support for older versions of Kubernetes and the many bug fixes in each patch release of Kubernetes, it is important to keep the versions up to date. Also, Kubernetes is new software and security vulnerabilities are announced on regular basis.

Transport Layer Security (TLS): It is critical that Kubernetes components are using an encrypted connection. Organizations should make sure that Kubernetes clusters have end-to-end TLS enabled.

Kubernetes RBAC and authentication: Kubernetes has several forms of access management, including role-based access control (RBAC). Users should make sure that RBAC is enabled and that users are assigned proper roles.

Network policies enabled: **A** Kubernetes network is flat. Every pod is given an IP address and can communicate with all other pods on its namespace. It is possible to use network policies to limit the interactions among pods and lock down this cross-communication.

Different clusters or namespaces based on the security profile: Namespaces can create logical clusters in a single physical cluster, but they only provide soft isolation. Organizations should not include vastly different services in the same cluster. Using different Kubernetes clusters reduces the potential for vulnerabilities to affect all systems. Also, large clusters with unrelated services become harder to upgrade, which violates the first item in the first point of this checklist.

Limit access to insecure API server ports: The API Server is the main means to communicate with Kubernetes components. Besides enabling authentication and RBAC, organizations should lock down all unsecure API server ports.

Limit access of pod creation for users: One of the most widely used vectors used for attacks of container management systems are the containers themselves. There are a few things users should do in order to make sure the containers used are secure.

First, they should limit who can create pods. Second, they should limit the use of unknown or unmaintained libraries. Organizations should also use a private container registry. Containers within

the private registry should use tagged container images and keep tagged images immutable.

Secure Dashboard: The Kubernetes Dashboard is a great utility for users, but it can also be a vector for malicious attacks. It is important to limit its exposure as much as possible. Three specific tactics users should employ include:

- Not exposing the Dashboard to the Internet
- Making sure that the Dashboard ServiceAccount is not open and accessible to users
- Configuring the login page and enabling RBAC

Important Resources

- Tracking Kubernetes Releases
- Tracking Kubernetes Enhancement Proposals
- Check if the Kubernetes distribution or installer has been through conformance testing
- CIS Security Benchmark tool
- YAML Templates

Written by **Chris Gaun,** *CNCF Ambassador and Product Manager at Mesosphere*
Chris Gaun is a CNCF ambassador and product manager for Mesosphere Kubernetes Engine. He has presented at Kubecon several times and has hosted over 40 Kubernetes workshops across the US and EU. He lives in Mississippi with his beautiful wife Jasmin, kids, and dog Panda.

Written by **Joe Thompson,** *Solutions Architect at Mesosphere*
Joe Thompson is a solutions architect at Mesosphere. Prior to Mesosphere, he worked at CoreOS and Red Hat (among others), providing practical solutions and training in for Kubernetes environments. He has presented at KubeCon twice, and frequently presents at the DC-area NoVa Kubernetes meetup. He lives in West Virginia with his wife, two daughters, two dogs, two cats, and one freeloading rescued mouse.

DZone
A DEVADA MEDIA PROPERTY

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300