DZone®
A DEVADA MEDIA PROPERTY

BROUGHT TO YOU IN PARTNERSHIP WITH D2IQ

Advanced Kubernetes

CHRIS GAUN
PRODUCT MANAGER OF KUBERNETES PLATFORM, D2IQ

CONTENTS

ABOUT KUBERNETES

Kubernetes is a distributed cluster technology that manages a container-based system in a declarative manner using an API. Kubernetes is an open-source project governed by the Cloud Native Computing Foundation (CNCF) and counts all the largest cloud providers and software vendors as its contributors along with a long list of community supporters.

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 group, operations needs to architect and manage both the core Kubernetes container orchestration and the necessary auxiliary solutions (addons) — for example, monitoring, logging, and CI/CD pipeline. The CNCF maps out many of these solutions and groups them by category in the "CNCF Landscape."

In 2019, Gartner predicted that by 2022, more than 75% of global organizations would be running containerized applications in production, up from less than 30% at that time. The CNCF Survey 2019, published roughly one year later in 2020, says that more than that are already in production. This research points out how quickly Kubernetes is being adopted and what a strategic role it plays in the global digital landscape.

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.

D2
IQ

The Leading Independent Platform for Enterprise Kubernetes

Explore the D2iQ Kubernetes Platform

D2iQ simplifies and automates the really difficult tasks needed for enterprise Kubernetes in production at scale.

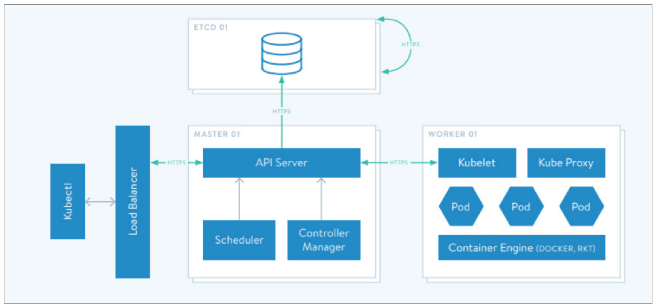
Simplify your Kubernetes Journey

- **Declarative** – Kubernetes' 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.

- **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

Figure 1: Simplified Kubernetes architecture and components



CONTROL PLANE, NODES, AND PERSISTENT STORAGE

Kubernetes' basic architecture requires a number of components.

API SERVER

The Kubernetes API server handles all requests coming into the cluster. Users, as well as other Kubernetes components, send events to the Kubernetes API Server through HTTPS (port 443). The API Server then processes events and updates the Kubernetes persistent data store, usually etcd. The API Server also performs authentication and authorization depending on how the cluster was configured.

https://kubernetes.io/docs/concepts/architecture/master-node-communication/

ETCD CLUSTER

Kubernetes uses etcd for persistent storage, providing a single source for all Kubernetes objects. A best practice is to ensure that there are multiple etcd instances to ensure high availability of the cluster. A loss of etcd storage will result in a loss of the cluster's state, so etcd should be backed up for disaster recovery. Since etcd is the single source of truth for the cluster, it's imperative to secure it against malicious actors.

https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/

CONTROLLER MANAGER

Kubernetes uses a control plane with several types of controllers to perform non-terminating control loops that observe the state of Kubernetes objects and reconcile it with the desired state. This includes a wide variety of functions such as invoking pod admission controllers, setting pod defaults, or injecting sidecar containers into pods, all according to the configuration of the cluster.

https://kubernetes.io/docs/concepts/architecture/controller/

KUBELET (AGENT)

Each node in a Kubernetes cluster has an agent called the Kubelet. The Kubelet manages the container runtime (e.g., Docker, containerD, rkt) on individual nodes. It is the component that enables orchestration of multiple containers across many nodes and is critical to any Kubernetes cluster. The logs it produces are often the first place to troubleshoot problems and can be viewed on any given node with the command `$ journalctl -u kubelet`.

https://kubernetes.io/docs/concepts/overview/components/

SCHEDULER

Kubernetes relies on a sophisticated algorithm to schedule Kubernetes objects. The scheduling takes into account attributes from a given workload (e.g., resource requirements) and applies custom prioritization settings (e.g., affinity rules) to provision pods on particular Kubernetes worker nodes.

https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

KUBE-PROXY

Each Kubernetes cluster worker node has a network proxy for connectivity to application services in the cluster. The proxy reflects the services defined in the cluster and enables simple stream and round-robin forwarding to a set of backends, regardless of which node they reside in the cluster.

https://kubernetes.io/docs/concepts/overview/components/#kube-proxy

OTHER KUBERNETES COMPONENTS (ADDONS)

To run the most basic Kubernetes cluster, a number of additional components and addons are required.

KUBE-DNS (COREDNS)

Production deployments of Kubernetes include a DNS server, which can be used for service discovery. All services and pods in a cluster are assigned a domain name that is resolved by the DNS. This DNS is ordinarily handled by Kubernetes DNS, which can be backed by popular services such as CoreDNS (coredns.io). With Kubernetes DNS configured, Services (and pods) can be addressed by a defined naming convention in the form of A/AAAA or SRV records for discernable access by clients or other Services in the cluster.

KUBECTL

The official command line for Kubernetes is called kubectl. 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. Second is the Kubernetes metrics API, which provides an API to access these aggregated metrics.

WEB UI (DASHBOARD)

Kubernetes has an official GUI called Dashboard. That GUI is distinct from 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.

CONSTRUCTS

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

CONSTRUCT	DESCRIPTION
Namespaces	• Kubernetes includes a means to segment a single physical cluster into separate logical clusters using namespacing. • Can be used to isolate users, teams, or applications and set quotas among other functions.
Pods	• Encapsulate one or more tightly bound containers, the resources required to run these containers, and a network namespace. • Always scheduled on a single node, regardless of the number of containers that are run as part of that pod.
StatefulSets	• A Kubernetes controller for managing workloads that require additional management of state due to application requirements. • Pods managed by a StatefulSet have a consistent naming convention and manner of connecting to persistent storage that might be required for some applications, particularly legacy apps or databases.
ReplicaSets	• A construct (ordinarily created by a Deployment object) that ensures the desired number of copies (or replicas) of a pod that share a particular container image(s) are running. • Replicas may be used in the background when performing rollouts on a new deployment.
Roles	• Sets of permissions that can be assigned to users or service accounts of a cluster. • Contain rules that specify which API set, object, and verbs are permitted for an account that's assigned a particular role.
Ingress and Load Balancing	• Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. • Load balancers ensure traffic is routed to multiple instances of the service running on the cluster to avoid a single point of failure of the service.
Deployments	• Construct that provides capabilities for managing to the desired state of an app, allowing higher availability and dynamic scaling of workloads. • Include container deployment, placement, scaling, image updates, rollout, and rollback of a workload.
Services	• Enable a consistent mechanism to access applications inside a cluster by providing a logical layer that assigns IP/DNS/etc. persistence to apps.
DaemonSet	• Enables users to run a pod on all nodes in the cluster. • Ordinarily used by Kubernetes plugins or administrative addons that require code to be executed throughout the cluster (e.g., log aggregation, networking features).
Jobs and Cronjobs	• Logic to run processes that run to completion (jobs) and processes that run at specific intervals to completion (cronjobs). • Kubernetes CronJobs schedule configuration is identical to Linux CronJob configuration.

TABLE CONTINUES IN NEXT COLUMN

EXTENSIONS

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

POINT	DESCRIPTION
Custom Resource Definition (CRD)	• Allows users to extend Kubernetes with custom APIs for different objects beyond the standard ones supported by Kubernetes.
Container Runtime Interface (CRI)	• A plugin API that enables Kubernetes to support other container runtimes beyond Docker and Containerd.
Container Network Interface (CNI)	• Gives users a choice of network overlay that can be used with Kubernetes to add networking features. • A networking plugin is required to run Kubernetes (e.g., Calico, Flannel, Canal, and other more niche/specific plugins).
Container Storage Interface (CSI)	• Empowers users to support different storage systems through a driver model.

KUBECTL

Below are some commands useful for IT professionals getting started with Kubernetes. A full list of Kubectl commands: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands

MAKING LIFE EASIER

Finding Kubernetes command short name:

```
$ kubectl describe
```

Find out more:

- Using Kubectl Aliases – https://github.com/ahmetb/kubectl-aliases
- Switching among Kubernetes clusters and namespaces (context switching) – https://github.com/ahmetb/kubectx
- Tailing logs from multiple pods – https://github.com/wercker/stern

Setting a Bash prompt to mirror Kubernetes context:

```
prompt_set() {
if [ "$KUBECONFIG" != "" ]; then
  PROMPT_KUBECONTEXT="k8s:$(kubectl config current-context 2>/dev/null)\n"
fi
PS1="${PROMPT_KUBECONTEXT}[\u@\h \W]\$ "
}
k8s:admin@local
[kensey@sleeper-service ~]$
```

COMMANDS

KUBECTL COMMAND	FORMAT
`$ kubectl version`	*Print the client and server version information*
`$ kubectl cluster-info`	*IP addresses of master and services*
`$ kubectl get namespaces`	*List all the namespaces in the Kubernetes cluster*
`$ kubectl cordon NODE`	*Mark node as unschedulable; used for cluster maintenance*
`$ kubectl uncordon NODE`	*Mark node as available for scheduling; used after maintenance*
`$ kubectl drain NODE`	*Remove pods from node via graceful termination for maintenance*
`$ kubectl drain NODE --dry-run=client`	*Find the names of the objects that will be removed*
`$ kubectl drain NODE --force=true`	*Remove pods even if they refuse to terminate gracefully*
`$ kubectl taint nodes node1 key=value:NoSchedule`	*Repel work so only pods with a matching "toleration" can run on the node* Pods with appropriate toleration include identically tainted nodes in the list of candidate nodes during scheduling; pods without the matching toleration don't.
`$ kubectl explain RESOURCE`	*Print documentation of resources* This is especially useful if you require information such as the API group of a particular construct, fields needed in the manifest, and what those fields do.
`$ kubectl scale --replicas=COUNT rs/foo`	*Scale a ReplicaSet (rs) named `foo` to a particular number of pods; scale a Replication Controller or StatefulSet*
`$ kubectl apply -f frontend-v2.json`	*Perform a rolling update (where v2 is the next version of the service, provided its object names are identical to the existing ones running in the cluster)*
`$ kubectl label pods foo GPU=true`	*Add label `GPU`; set value to `true` to the pods named `foo`* If the label `GPU` already exists on the pod `foo`, the command will fail unless you add the flag `--overwrite`.
`$ kubectl delete pod foo`	*Delete `foo` pods from cluster*
`$ kubectl delete svc foo`	*Delete the service called `foo` from cluster*
`$ kubectl create service clusterip foo`	*Create a clusterIP for a service named `foo`*
`$ kubectl autoscale deployment foo --min=2 --max=10 --cpu-percent=70`	*Autoscale pod `foo` with a minimum of 2 and maximum of 10 replicas when CPU utilization is equal to or greater than 70%*

TABLE CONTINUES IN NEXT COLUMN

MANAGE NODES

Sometimes it's necessary to perform maintenance on underlying nodes. In those cases, it's important to use eviction and ensure the application owners have set a pod disruption budget. To properly evict running pods from a node that requires maintenance, use:

1. **Cordon node:** `$ kubectl cordon $NODENAME`
2. **Drain node:** `$ kubectl drain $NODENAME --ignore-daemonsets`
   - Automatically cordons the node (removes it from the possible schedulable node pool)

  - Respects `PodDisruptionBudgets`
  - Will not error out for running DaemonSet pods

3. Uncordon node: `$ kubectl uncordon $NODENAME`

TEST PLAN

It's important to devise a test plan and a run book to ensure any new clusters meet operational guidelines. This enables standard operating procedures (SOPs) that address needs such as consistency or compliance, facilitating better operational metrics for uptime and availability. Below are samples from D2iQ's Kubernetes test plan, which is used when validating new clusters, Kubernetes versions, addons, or workloads running on the cluster. 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 the 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 different provider-specific load balancer features
7. Expose a service using Ingress

TROUBLESHOOT

TROUBLESHOOT WITH KUBECTL

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

CHECK PERMISSIONS

	COMMAND	EXAMPLE
Check permissions	`$ kubectl auth can-i`	`$ kubectl auth can-i create deployments --namespace dev`
Check permission-specific user	`$ kubectl auth can-i [Options]`	`$ kubectl auth can-i create deployments --namespace dev --user=chris`

PENDING AND WAITING PODS

Pending —> Check Resources

TROUBLESHOOTING	COMMAND
General issue checking	`$ kubectl describe pod <name of pending pod>`
Check if pod is using too many resources (only if Kubernetes Metrics API is available)	`$ kubectl top pod`
Check node resources (only if Kubernetes Metrics API is available)	`$ kubectl top node`
Get node resources	`$ kubectl get nodes -o yaml | grep -A 15 allocatable`
Get all pod resources	`$ kubectl top pod --all-namespaces --containers=true`

Remove pods from the environment.

Waiting —> Incorrect Image Information

TROUBLESHOOTING	COMMAND

Check YAML URI	Look at the appropriate manifest file; check the image URI:
	`spec:`
	`  containers:`
	`  - name: example`
	`    image: url:port/image:v`

Pull an image onto desktop to troubleshoot	`docker pull <image>`

Also check that the secret information is correct.

CRASH LOOPING DEPLOYMENT

- Roll back deployment
- Troubleshoot

TROUBLESHOOTING	COMMAND
General issue checking	`$ kubectl describe deployments`
Roll back deployment to last version	`$ kubectl rollout undo [Deployment Name]`
Roll back deployment to specific version	`$ kubectl rollout undo [Deployment Name] --to-revision [number of revision]`
Pause a deployment	`$ kubectl rollout pause [Deployment Name]`

GENERAL DEBUGGING

Watch the `stdout` of a container:

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

Launch a temporary pod with an interactive shell:

```
$ kubectl run busybox --rm -i --tty --restart=Never
--image busybox -- /bin/sh
```

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

```
$ kubectl cp default/bash-rand-77d55b86c7-bntxs:/
root/rand .
tar: Removing leading `/' from member names

$ cat rand
567bea045d8b80cd6d007ced02849ac4
```

Increase the verbosity of `kubectl` output (99 is the most verbose — “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
```

CODE CONTINUES IN NEXT COLUMN

```
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
```

Get more information about Kubernetes resources:

```
$ 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:
[...]
```

Control output:

```
$ 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
[...]
```

TROUBLESHOOT WITH JQ

Kubectl provides an option to output results in JSON format, which allows for additional parsing and filtering. This is beneficial when writing scripts or monitoring the state of resources in a Kubernetes cluster. A popular open-source utility called `jq` makes it easy to parse JSON from the command line. Instead of writing code to parse the JSON objects, the output can be piped into the `jq` utility to filter out resources that meet certain criteria. For example, `kubectl` can be used to print out all pods in json format, but `jq` adds value by parsing out only pods with a specific start date or container image.

Basic usage:

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

Use the `kubectl json` format and pipe to the `jq` utility using a simple dot filter to pretty-print the output:

```
$ kubectl get nodes -o json | jq '.'
```

Use the hierarchy to filter the input so only node labels are displayed:

```
$ kubectl get nodes -o json | jq '.items[].
metadata.labels'
```

ADVANCED FUNCTIONS

Find pods with containers that have not passed a readiness probe:

- `select` – cherry-pick a piece of the input by criteria and use a boolean expression to match desired value
- Pipe the output of one filter into another filter to clean up the results

```
 $ kubectl get pods -o json | jq '.items[] | select(
.status.containerStatuses[].ready == false ) |
.metadata.name'
```

TROUBLESHOOT 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.

```
$ curl --header "Authorization: Bearer [token]"
[API server URL]
```

Build and submit requests:

- `GET` – Request is contained in the URL itself (default method) and 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": [
```

FEED RESULTS TO EXTERNAL SCRIPTS

`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"
 }
 [...]
```

`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

 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
```

ETCD

A production Kubernetes cluster will typically have three etcd nodes, which provides high availability and can withstand the loss of a single etcd node. The `etcdctl` utility provides a number of commands for running operations on the etcd cluster:

DESCRIPTION	COMMAND
Set an environment variable for the `etcdctl` binary, endpoints, and the required certificates and key	`ETCDCTL="ETCDCTL_API=3 etcdctl --cert=server.crt --key=server.key --cacert=ca.crt --endpoints=https://127.0.0.1:2379"`

TABLE CONTINUES ON NEXT PAGE

List members of the etcd cluster	`$ ETCDCTL member list`
Check the etcd endpoint health	`$ ETCDCTL endpoint health --cluster -w table`
Check the etcd endpoint status	`$ ETCDCTL endpoint status --cluster -w table`
List any active alarms	`$ ETCDCTL alarm list`
Create an etcd backup	`$ ETCDCTL snapshot save snapshot.db`

SECURITY CHECKLIST

Security is fundamental for Kubernetes day-two operations. Users should maintain basic security for the clusters.

Here is a security checklist for Kubernetes administrators:

☐ **Kubernetes patch releases** – i.e., 1.x.y (ensure it has all CVE fixes)

A new version of Kubernetes is released every three months. Patches come out at regular intervals for the latest three versions and have bug fixes.

Versions must be kept up to date due to the lack of widespread support for older versions of Kubernetes and the bug fixes in each patch release. *Note: Kubernetes constantly releases security fixes, and unpatched clusters are vulnerable.*

☐ **Transport Layer Security (TLS)**

Kubernetes components must use an encrypted connection. 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). 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 that 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. Do 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 this checklist.

☐ **Implement resource quotas**

To prevent "noisy neighbors" and potential denial of service situations, do not let containers run without an upper bound on resources. By default, all resources are created with unbounded CPU and memory limits.

To avoid the default behavior, assign resource quotas at the namespace level, which are critical in preventing overconsumption of resources.

☐ **Limit access to insecure API server ports**

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

☐ **Limit access of pod creation for users**

One of the most widely used vectors for attacking container management systems are the containers themselves. To ensure the containers are secure and protect the system from attacks:

- Limit who can create pods.
- Limit the use of unknown or unmaintained libraries.
- Use a private container registry and tagged container images, keeping tagged images immutable.

☐ **Secure Dashboard**

The Kubernetes Dashboard is a functional utility for users, but it can also be a vector for malicious attacks — limit its exposure as much as possible.

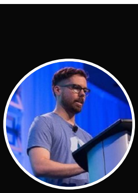
Employ three specific tactics:

- Do not expose the Dashboard to the Internet
- Ensure the Dashboard ServiceAccount is not open and accessible to users
- Configure the login page and enable RBAC

ADDITIONAL RESOURCES

- Tracking Kubernetes Releases: https://github.com/kubernetes/sig-release/tree/master/releases
    - Enhancements, Bug Triage, and Release Calendar
- Tracking Kubernetes Enhancement Proposals: https://github.com/kubernetes/enhancements/tree/master/keps
- Check if the Kubernetes distribution or installer has been through conformance testing: https://docs.google.com/spreadsheets/d/1LxSqBzjOxfGx3cmtZ4EbB_BGCxT_wlxW_xgHVVa23es/edit#gid=0
- Run end-to-end tests to validate a Kubernetes cluster: https://github.com/kubernetes/test-infra
- Kubernetes declarative test tool (KUTTL): https://github.com/kudobuilder/kuttl
- CIS Security Benchmark tool: https://github.com/mesosphere/kubernetes-security-benchmark
- YAML Templates: https://cheatsheet.dennyzhang.com/kubernetes-yaml-templates
- Use jq to process JSON output from the command line: https://stedolan.github.io/jq/

WRITTEN BY CHRIS GAUN,
PRODUCT MANAGER OF KUBERNETES PLATFORM, D2IQ



Chris Gaun is a CNCF ambassador and product manager for the D2iQ Kubernetes Platform. 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.

CONTRIBUTIONS BY CORBIN PACHECO,
PRODUCT AND TECHNICAL MARKETING, D2IQ

Corbin Pacheco leads the Product and Technical Marketing Team at D2iQ and has been involved in the pioneering of the Cloud Native Ecosystem since 2016. He has worked with and advised some of the leading Enterprises around the world in the adoption of Kubernetes and Cloud Native solutions.

DZone, a Devada Media Property, is the resource software developers, engineers, and architects turn to time and again to learn new skills, solve software development problems, and share their expertise. Every day, hundreds of thousands of developers come to DZone to read about the latest technologies, methodologies, and best practices. That makes DZone the ideal place for developer marketers to build product and brand awareness and drive sales. DZone clients include some of the most innovative technology and tech-enabled companies in the world including Red Hat, Cloud Elements, Sensu, and Sauce Labs.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC 27709

888.678.0399 919.678.0300