

# **WA3003 Docker and Kubernetes Administration Training**



Web Age Solutions Inc.  
USA: 1-877-517-6540  
Canada: 1-877-812-8887  
Web: <http://www.webagesolutions.com>

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

IBM, WebSphere, DB2 and Tivoli are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

For customizations of this book or other sales inquiries, please contact us at:

USA: 1-877-517-6540, email: [getinfousa@webagesolutions.com](mailto:getinfousa@webagesolutions.com)

Canada: 1-877-812-8887 toll free, email: [getinfo@webagesolutions.com](mailto:getinfo@webagesolutions.com)

Copyright © 2022 Web Age Solutions Inc.

This publication is protected by the copyright laws of Canada, United States and any other country where this book is sold. Unauthorized use of this material, including but not limited to, reproduction of the whole or part of the content, re-sale or transmission through fax, photocopy or e-mail is prohibited. To obtain authorization for any such activities, please write to:

Web Age Solutions Inc.  
821A Bloor Street West  
Toronto  
Ontario, M6G 1M1

# Table of Contents

Chapter 1 - Docker and Linux Container Technology: Introduction and Use-Cases.....	11
1.1 Modern Infrastructure Terminology.....	11
1.2 Virtualization.....	12
1.3 Hypervisors.....	12
1.4 Hypervisor Types.....	12
1.5 Type 1 Hypervisors.....	13
1.6 Type 2 Hypervisors.....	13
1.7 Type 1 vs Type 2 Processing.....	14
1.8 Paravirtualization.....	15
1.9 Virtualization Qualities (1/2).....	15
1.10 Virtualization Qualities (2/2).....	15
1.11 Disadvantages of Virtualization.....	16
1.12 Containerization.....	16
1.13 Virtualization vs Containerization.....	17
1.14 Where to Use Virtualization and Containerization.....	17
1.15 Containerization: High-Level.....	17
1.16 Popular Containerization Systems.....	18
1.17 What are Linux Containers.....	18
1.18 Docker.....	19
1.19 OpenVZ.....	19
1.20 Solaris Zones (Containers).....	20
1.21 Container Orchestration Tools.....	20
1.22 Docker Swarm.....	20
1.23 Kubernetes.....	21
1.24 Mesos and Marathon.....	22
1.25 Mesos and Marathon (contd.).....	22
1.26 Docker Use-Cases.....	23
1.27 Microservices.....	23
1.28 Microservices and Containers / Clusters.....	23
1.29 Summary.....	24
Chapter 2 - Docker in Action.....	25
2.1 Docker Basics.....	25
2.2 Where Can I Run Docker?.....	26
2.3 Installing Docker Container Engine.....	26
2.4 Docker Toolbox.....	27
2.5 What is Docker?.....	27
2.6 Docker Architecture.....	28
2.7 Docker Architecture Diagram.....	28
2.8 Docker Images.....	29
2.9 Docker Containers.....	29
2.10 Docker Integration.....	30
2.11 Docker Services.....	30
2.12 Docker Application Container Public Repository.....	31
2.13 Docker Run Command.....	31

2.14 Starting, Inspecting, and Stopping Docker Containers.....	31
2.15 Docker Volume.....	32
2.16 Dockerfile.....	32
2.17 Docker Compose.....	33
2.18 Using Docker Compose.....	33
2.19 Dissecting docker-compose.yml.....	34
2.20 Specifying services.....	34
2.21 Dependencies between containers.....	35
2.22 Injecting Environment Variables.....	35
2.23 Summary.....	36
Chapter 3 - Managing Docker State.....	37
3.1 State and Data in Docker.....	37
3.2 Volumes.....	38
3.3 More About Volumes.....	38
3.4 Uses for Volumes.....	38
3.5 Working With Volumes.....	39
3.6 Create Volume.....	39
3.7 Use Volumes with Containers.....	40
3.8 Bind Mounts.....	40
3.9 Using Bind Mounts.....	41
3.10 tmpfs Mounts.....	41
3.11 Storing Data in the Container.....	42
3.12 Storage Drivers.....	43
3.13 Remote Data Storage.....	43
3.14 Networking.....	44
3.15 The Default Bridge Network.....	45
3.16 User-Defined Bridge Networks.....	45
3.17 Docker Network Commands.....	46
3.18 Creating a User-Defined Bridge Network.....	46
3.19 Summary.....	47
Chapter 4 - Open Container Initiative and Container Runtime Interface.....	49
4.1 Open Container Initiative (OCI).....	49
4.2 Docker.....	49
4.3 Docker Engine Architecture.....	50
4.4 runC.....	50
4.5 containerd.....	51
4.6 containerd Benefits.....	51
4.7 CRI-O.....	52
4.8 CRI-O Components.....	52
4.9 Kubernetes and CRI-O.....	53
4.10 Using Container Runtimes with Minikube.....	53
4.11 Docker Runtime and Kubernetes.....	53
4.12 Putting Things Together.....	54
4.13 Summary.....	55
Chapter 5 - Kubernetes Architecture.....	57
5.1 Kubernetes Basics.....	57

5.2 What is Kubernetes?	57
5.3 Container Orchestration	58
5.4 Architecture Diagram	58
5.5 Components	59
5.6 Kubernetes Cluster	59
5.7 Master Node	60
5.8 Kube-Control-Manager	60
5.9 Nodes	61
5.10 Pod	61
5.11 Using Pods to Group Containers	62
5.12 Label	62
5.13 Label Syntax	63
5.14 Label Selector	63
5.15 Annotation	64
5.16 Persistent Storage	64
5.17 Resource Quota	64
5.18 Interacting with Kubernetes	65
5.19 Summary	65
Chapter 6 - Working with Kubernetes	67
6.1 Installation	67
6.2 Startup	68
6.3 Kubernetes Tools	68
6.4 kubectl Command Line Interface	68
6.5 API Proxy	69
6.6 Dashboard	70
6.7 Kubernetes Component Hierarchy	70
6.8 Deployments	70
6.9 Deployment Commands	71
6.10 Updating Deployments	72
6.11 Network Considerations	73
6.12 Services	73
6.13 Namespaces	74
6.14 Labels	75
6.15 Annotations	75
6.16 Other Useful Commands	76
6.17 Summary	76
Chapter 7 - Kubernetes Workload	79
7.1 Kubernetes Workload	79
7.2 Kubernetes Workload (contd.)	80
7.3 Managing Workloads	80
7.4 Imperative commands	80
7.5 Imperative Object Configuration	81
7.6 Declarative Object Configuration	81
7.7 Configuration File Schema	82
7.8 Understanding API Version	82
7.9 Understanding API Version	83

7.10	Obtaining API Versions.....	83
7.11	Obtaining API Versions (contd.).....	84
7.12	Stateless Applications.....	84
7.13	Sample Deployment Manifest File.....	85
7.14	Working with Deployments.....	85
7.15	Stateful Applications.....	86
7.16	Sample Stateful Manifest File.....	86
7.17	Sample Stateful Manifest File (Contd.).....	87
7.18	Working with StatefulSet.....	87
7.19	Jobs.....	88
7.20	Sample Job Manifest File.....	88
7.21	Sample Job Manifest File (Contd.).....	89
7.22	Working with Batch Job.....	89
7.23	DaemonSets.....	90
7.24	DaemonSets (contd.).....	90
7.25	Sample Daemon Manifest File.....	91
7.26	Rolling Updates.....	91
7.27	Rolling Updates (Contd.).....	92
7.28	Rolling Updates (Contd.).....	92
7.29	Summary.....	93
Chapter 8 - Scheduling and Node Management.....		95
8.1	Kubernetes Scheduler.....	95
8.2	Kubernetes Scheduler Overview (contd.).....	96
8.3	Kubernetes Scheduler Overview (contd.).....	96
8.4	Skip Kubernetes Scheduler.....	97
8.5	Scheduling Process.....	97
8.6	Scheduling Process - Predicates.....	97
8.7	Scheduling Process - Priorities.....	98
8.8	Scheduling Algorithm.....	98
8.9	Kubernetes Scheduling Algorithm.....	99
8.10	Scheduling Conflicts.....	99
8.11	Controlling Scheduling.....	100
8.12	Label Selectors.....	100
8.13	Label Selectors (contd.).....	101
8.14	Label Selectors (Contd.).....	101
8.15	Node Affinity and Anti-affinity.....	102
8.16	Node Affinity Example.....	102
8.17	Node Antiaffinity Example.....	103
8.18	Taints and Tolerations.....	103
8.19	Taints and Tolerations (Contd.).....	104
8.20	Taints and Tolerations (Contd.).....	104
8.21	Taints and Tolerations - Example.....	104
8.22	Summary.....	105
Chapter 9 - Managing Networking.....		107
9.1	Kubernetes Networking Components.....	107
9.2	The Kubernetes Network Model.....	107

9.3 Networking Scenarios.....	108
9.4 Container-Container Communication.....	108
9.5 Pod-Pod Communication.....	109
9.6 1.3 Pod-Service Communication.....	109
9.7 External-Service Communication.....	109
9.8 Accessing Applications.....	110
9.9 Useful Commands.....	111
9.10 Container Network Interface (CNI).....	111
9.11 What is CNI's Role?.....	112
9.12 CNI Configuration Format.....	112
9.13 Sample CNI Configuration.....	112
9.14 Running the CNI Plugins.....	113
9.15 Summary.....	114
Chapter 10 - Managing Persistent Storage.....	115
10.1 Storage Methods.....	115
10.2 Container OS file system storage.....	115
10.3 Docker Volumes.....	116
10.4 Kubernetes Volumes.....	116
10.5 K8S Volume Types.....	117
10.6 Cloud Resource Types.....	117
10.7 configMaps.....	118
10.8 Creating configMaps from Literals.....	118
10.9 Creating configMaps from files.....	119
10.10 Using configMaps.....	119
10.11 emptyDir.....	120
10.12 Using an emptyDir Volume.....	120
10.13 Other Volume Types.....	121
10.14 Persistent Volumes.....	122
10.15 Creating a Volume.....	122
10.16 Persistent Volume Claim.....	123
10.17 Persistent Volume.....	123
10.18 Pod that uses Persistent Volume.....	125
10.19 Secrets.....	125
10.20 Creating Secrets from Files.....	126
10.21 Creating Secrets from Literals.....	126
10.22 Using Secrets.....	127
10.23 Security Context.....	127
10.24 Security Context Usage.....	128
10.25 Summary.....	129
Chapter 11 - Working with Helm.....	131
11.1 What is Helm?.....	131
11.2 Installing Helm.....	132
11.3 Helm and KUBECONFIG.....	133
11.4 Helm Features.....	133
11.5 Helm Terminology.....	134
11.6 Searching for Charts with helm CLI.....	134

11.7 Adding Repositories.....	135
11.8 Helm Hub - Search.....	135
11.9 Helm Hub - Chart Page.....	136
11.10 Installing a Chart.....	136
11.11 Upgrading a Release.....	137
11.12 Rolling Back a Release.....	137
11.13 Creating Custom Charts.....	138
11.14 Common Chart Files.....	138
11.15 Helm Templates.....	139
11.16 Installing A Custom Chart.....	140
11.17 Packaging Custom Charts.....	140
11.18 Summary.....	141
Chapter 12 - Logging, Monitoring, and Troubleshooting.....	143
12.1 Differences Between Logging and Monitoring.....	143
12.2 Logging in Kubernetes.....	143
12.3 Basic Logging.....	144
12.4 Logging Agents.....	144
12.5 Fluentd and Elastic Stack.....	145
12.6 Monitoring with Prometheus.....	146
12.7 Kubernetes and Prometheus - Metrics.....	146
12.8 Alerting.....	147
12.9 Debugging Pods.....	147
12.10 Debugging Pods (Contd.).....	148
12.11 Debugging Nodes.....	148
12.12 Debugging Replication Controllers and Services.....	148
12.13 Upgrading Kubernetes.....	148
12.14 Upgrade Process.....	149
12.15 Determine Which Version to Upgrade To.....	150
12.16 Upgrade kubeadm.....	150
12.17 Upgrade Control Plane Node.....	150
12.18 Upgrade kubelet and kubectl.....	151
12.19 Upgrade Worker Nodes.....	151
12.20 Recovering From a Failure State.....	152
12.21 Summary.....	152
Chapter 13 - Continuous Integration Fundamentals.....	153
13.1 Jenkins Continuous Integration.....	153
13.2 Jenkins Features.....	153
13.3 Running Jenkins.....	154
13.4 Downloading and Installing Jenkins.....	154
13.5 Running Jenkins as a Stand-Alone Application.....	155
13.6 Running Jenkins on an Application Server.....	156
13.7 Installing Jenkins as a Windows Service.....	156
13.8 Different types of Jenkins job.....	157
13.9 Configuring Source Code Management(SCM).....	158
13.10 Working with Subversion.....	159
13.11 Working with Subversion (cont'd).....	160



13.12	Working with Git.....	160
13.13	Build Triggers.....	161
13.14	Schedule Build Jobs.....	162
13.15	Polling the SCM.....	163
13.16	Maven Build Steps.....	163
13.17	Configuring Jenkins to Access Kubernetes.....	164
13.18	Jenkins Pipeline.....	164
13.19	Jenkins Pipeline Output.....	165
13.20	Installing Jenkins Plugins.....	165
13.21	Summary.....	166



# Chapter 1 - Docker and Linux Container Technology: Introduction and Use-Cases

---

## *Objectives*

Key objectives of this chapter

- Infrastructure Terminology
- Virtualization
- Containers
- Container Orchestration
- Docker Use-Cases
- Microservices

## 1.1 Modern Infrastructure Terminology

- **Container** - A unit of software that bundles an application's code and its dependencies, providing reliable execution and portability. **Docker** is a container technology. The capabilities of Container are often compared with those of systems like VMWare and VirtualBox which provide OS-level virtualization.
- **Cluster** - A group of servers providing a single service that work together to support fail-over and high availability. **Kubernetes** provides clustering capability for Docker containers.
- **Cloud** - A non-local infrastructure that supports application execution, scaling and geographical distribution. Amazon AWS and Google App Engine are examples of cloud infrastructures.

## Notes

Clusters, containers, and clouds: understanding modern infrastructure

Container lifecycles and challenges

Local versus production: bridging development environments

## 1.2 Virtualization

- Virtualization is a technique of abstracting computer resources (hardware and networking) and allowing software (OSes and applications) to run in such environments as if it were a real computer
- The collection of the virtualized resources are packaged in a Virtual Machine
- Virtual Machines are managed by hypervisors (Virtual Machine Managers)
- Virtualization helps drive server consolidation initiatives that lead to better resource utilization, management and driving operational costs down in public cloud computing, on-premise private clouds, and traditional enterprise data centers
- Virtualization was first introduced in mainframes back in the 70s

## 1.3 Hypervisors

- A hypervisor (or Virtual Machine Monitor (VMM)) is a specialized software program that creates and runs virtual machines
- Hypervisors allow several operating systems (called "Guest OSes") to share a single hardware host at the same time in a way that
  - ◇ Each guest OS receives its own fair share of virtualized resources (CPU, RAM, network, file-system, etc.)
  - ◇ Guest OSes running on the same host do not affect others (allowing for a safe multi-tenant hosting arrangement)

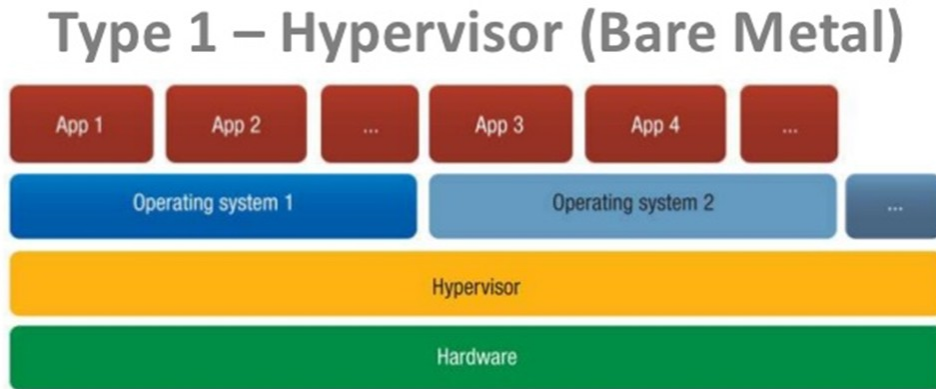
## 1.4 Hypervisor Types

- There are two types of hypervisors:
  - ◇ Type 1 (native) hypervisors
  - ◇ Type 2 (hosted) hypervisors

### Notes

## 1.5 Type 1 Hypervisors

- Type 1 (native) hypervisors run directly on the host's hardware and manage guest OSes that are executed just a level above the hypervisor



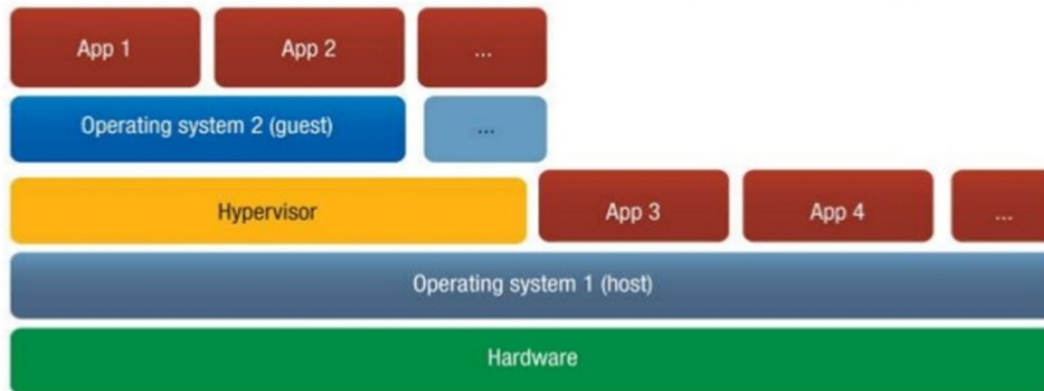
- ◇ Examples of Type 1 hypervisors: the Citrix XenServer, VMware ESX/ESXi, KVM, Microsoft Hyper-V, Oracle VM Server for SPARC, Oracle VM Server for x86 hypervisor

### Notes

## 1.6 Type 2 Hypervisors

- Type 2 (hosted) hypervisors run as a system program on a regular operating system environment (FreeBSD, Linux, or Windows.). The guest OSes run on top of the hypervisor

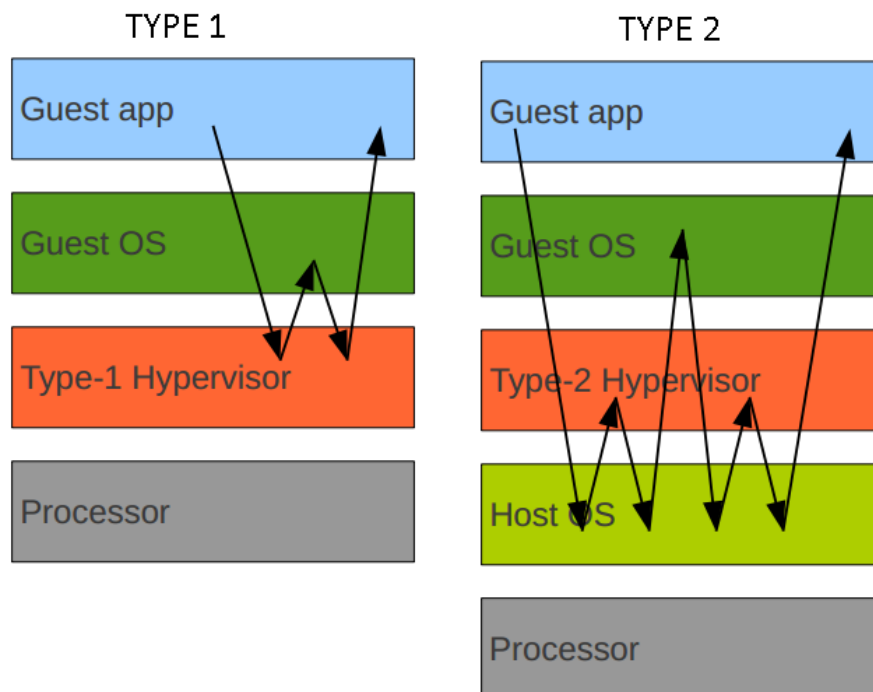
## Type 2 – Hypervisor (Hosted)



- ◇ Examples of Type 2 hypervisors: VMware Workstation and VirtualBox are examples of Type 2 hypervisors

### 1.7 Type 1 vs Type 2 Processing

- Processing occurs differently between these two basic types of VM. Type 1 relies upon native hardware that supports virtualization whereas Type 2 introduces a software abstraction layer to facilitate the virtualization



## 1.8 Paravirtualization

- Paravirtualization (PV) is a special case of system virtualization
- With PV, a hardware environment is not entirely simulated and the guest OSES get coordinated access to the underlying physical hardware bypassing the VMM in certain cases
- PV requires the guest OSES to be modified (ported for the PV-API )
- A popular PV hypervisor *Xen* is developed under the Xen project using the GNU GPL v2 license (<http://www.xenproject.org/>)
- PV hypervisors have a simple design and offer excellent execution performance
  - ◇ For example, with Xen you get performance degradation between 2% for a standard workload and 10 – 20% for a worst-case scenario

## 1.9 Virtualization Qualities (1/2)

- Transparent sharing of resources (memory, network, disk, CPU, etc.)
  - ◇ decouples hardware from software
  - ◇ hardware aggregates as a pool of sharable resources
- Live migration
  - ◇ shift virtual servers between physical hardware instances while running
  - ◇ facilitate zero downtime while still maintaining hardware
- Isolation
  - ◇ limits security exposure
  - ◇ reduces the spread of risk

## 1.10 Virtualization Qualities (2/2)

- Management
  - ◇ single point of control across all VMs
  - ◇ ease deployment burden through repetitive scripts and templates

- ◇ block-level rollbacks to prior snapshots in the event of failure
- High availability
  - ◇ boot virtual servers on alternate hardware in the event of primary hardware failure
  - ◇ execute multiple instances of a virtual server across multiple hosts

### 1.11 Disadvantages of Virtualization

- System virtualization tools or emulators need to load and boot up a full image of the guest OS and, basically, need to emulate a complete machine, which results in a high operational overhead
- The virtualization overhead sharply increases in proportion to the number of guest OSes running on the same physical host:
  - ◇ CPU (CPU caches efficiencies fall)
  - ◇ Memory (swapping increases)
  - ◇ IO / Networking (contention increases)
  - ◇ The underlying scheduler contention

### Notes

### 1.12 Containerization

- Containerization is a lightweight alternative to full machine virtualization
  - ◇ It is often called a virtualization environment, or OS-level virtualization
- A container encapsulates an application running inside its own operating environment which is derived from the underlying host OS
- Containerization has been popularized by cloud-like processing environments that require fast server boot-up time
- At the moment, the most widely used technology behind containerization is **LinuX Containers (LXC)**, which is a userspace interface for the Linux



kernel containment features (cgroups and namespaces)

- The main containerization action is happening in the Linux space (and x84 (32-bit) & x64 (64-bit) machine architecture)

### **1.13 Virtualization vs Containerization**

- Both offer multi-tenancy for guests (OSes and applications)
- Virtualization is about translating communication between the hosted OSes and hypervisor
- Containerization is "native" in that containers share the host OS's kernel
  - ◇ Containers' OS is the same as the hosts' OS
- Virtualization allows running of multiple guest OSes on the same host, while containerization is limited to the OS type the host uses
- Traditional virtualization offers better protection from "rogue" tenants

### **1.14 Where to Use Virtualization and Containerization**

- Virtualization belongs to the Enterprise space (and big Ops budgets)
- If you are an infrastructure provider and a Linux shop competing on price, use containerization
  - ◇ Platform-as-a-Service (PaaS) vendors such as Heroku, OpenShift, and Cloud Foundry use Linux containerization
- Containerization is the way to go if you are a cost-conscious organization trying to save every nickel (most Linux containerization systems are free)
- DevOps can hugely benefit from containerization as well (fast system provisioning: build, test, integration machines, etc.)

### **1.15 Containerization: High-Level**

- Containerization is an OS-level lightweight alternative to traditional virtualization offering uses the following benefits:
  - ◇ Better runtime performance and system boot-up times

- ◇ Overall cheaper solutions (most containerization systems and tools are free)
- ◇ Container is an excellent choice if you are an infrastructure service provider of Linux-based systems
- Containerization has some disadvantages:
  - ◇ Relatively low protection against "rogue" tenants
  - ◇ Containers launched on a host will inherit the host's OS, which limits your choices
  - ◇ Currently very popular only in the Linux world

## Notes

### 1.16 Popular Containerization Systems

- Linux containerization systems:
  - ◇ **LXC** (more on LXC a bit later ...)
  - ◇ **Docker** (more on Docker a bit later ...)
  - ◇ **OpenVZ** (more on OpenVZ a bit later ...)
  - ◇ **Linux-VServer**
- Non-Linux systems containerization systems:
  - ◇ **Oracle Solaris Zones (Containers)** (more on them a bit later ...)
  - ◇ **IBM AIX Workload Partitions**
  - ◇ **FreeBSD Jails**

### 1.17 What are Linux Containers

- Linux Containers (LXC) is an OS-level virtualization environment that allows multiple Linux systems to run on a single physical host
- LXC provides a user API and a toolset for interfacing with the Linux kernel

containment features (cgroups and namespaces)

- The main differentiating factor between LXC and some other systems is that it runs on the unmodified Linux kernel of various Linux distros
- The LXC system is written in a number of programming languages, including C, Python 3, Lua, and others
- The initial release was in August 2008

## 1.18 Docker

- Docker is an open-source system for creating virtual environments
- Runs on any modern-kernel Linux distributions
- To communicate with the underlying host kernel, Docker uses virtualization interfaces based on a variety of systems:
  - ◇ *libvirt*
  - ◇ **LXC (LinuX Containers)**
  - ◇ *systemd-nspawn*
- You can install Docker inside a VirtualBox and run it on OS X or Windows
- Docker can be booted from the small footprint Linux distribution *boot2docker*
- Written in the Go programming language

## 1.19 OpenVZ

- OpenVZ is a Linux OS-level virtualization technology that leverages Linux modern features kernel
- OpenVZ is similar to FreeBSD jails and Oracle Solaris Zones
- Requires Linux kernel patching
  - ◇ As of version 4.0, OpenVZ can work with unpatched Linux 3.x kernels offering a reduced functionality
- Written in C

## **1.20 Solaris Zones (Containers)**

- Oracle Solaris Zones (previously known as Solaris Containers) is an OS-level virtualization technology for Intel-based and SPARC systems
- Oracle Solaris Zones are an integral part of the Oracle Solaris 10 and up
- Oracle promotes this technology as a way to maintain the one-application-per-server deployment model that share the same hardware resources
- Also available in operating systems based on the Illumos kernel, which is derived from OpenSolaris
  - ◇ e.g. SmartOS, OmniOS, OpenIndiana
- The first public release was in February 2004

## **1.21 Container Orchestration Tools**

- A typical enterprise application involves multiple applications or components.
- When we use microservices principles, we deploy each microservice in a separate container
- A container orchestration solution provides management features, such as load-balancing and fault-tolerance
- Container orchestration tools include:
  - ◇ Docker Swarm
  - ◇ Kubernetes
  - ◇ Mesos and Marathon

## **1.22 Docker Swarm**

- Docker's official container orchestration tool
- Uses the standard Docker API and networking, making it easy to drop into an environment where you're already working with Docker containers.
- It's recommended for smaller scale deployments

- Key concepts:
  - ◇ **Managers:** distribute tasks across the cluster, with one manager orchestrating the worker nodes that make up the swarm.
  - ◇ **Workers:** run Docker containers assigned to them by a manager.
  - ◇ **Services:** an interface to a particular set of Docker containers running across the swarm.
  - ◇ **Tasks:** the individual Docker containers running the image, plus commands, needed by a particular service.
  - ◇ **Key-value store:** etcd, Consul or Zookeeper storing the swarm's state and providing service discoverability.

## 1.23 Kubernetes

- Kubernetes is based on Google's experience of running workloads at a huge scale in production over the past 15 years.
- Kubernetes is recommended for medium to large scale deployments
- Key concepts:
  - ◇ **Master:** by default, a single master handles API calls, assigns workloads and maintains configuration state.
  - ◇ **Minions:** the servers that run workloads and anything else that's not on the master.
  - ◇ **Pods:** units of computing power, made-up of one or a handful of containers deployed on the same host, that together perform a task, have a single IP address and flat networking within the pods.
  - ◇ **Services:** front end and load balancer for pods, providing a floating IP for access to the pods that power the service, meaning that changes can happen at the pod-level while maintaining a stable interface.
  - ◇ **Replication controllers:** responsible for maintaining X replicas of the required pods.
  - ◇ **Labels:** key-value tags (e.g. "Name") that you and the system used to identify pods, replication controllers and services.

## Notes

### 1.24 Mesos and Marathon

- Apache Mesos is a cluster manager that makes computing resources available to frameworks
- It provides similar functionality to Docker/Kubernetes which it predates.
- The frameworks deal with the specifics of what runs on the Mesos cluster.
- Marathon is one such framework and it specializes in running applications, including containers, on Mesos clusters.
- Together, Mesos and Marathon offer an equivalent to Kubernetes, while also allowing you to run non-containerized workloads alongside containers.
- It is recommended for large scale scenarios and also when you need to run non-containerized workloads alongside containers

### 1.25 Mesos and Marathon (contd.)

- Key concepts:
  - ◇ **Masters:** Zookeeper manages a minimum of three master nodes and enables high availability by relying on a quorum amongst those nodes.
  - ◇ **Slaves:** these nodes run the tasks passed down by the framework.
  - ◇ **Framework:** Mesos itself knows nothing about the workloads, whereas specialist frameworks decide what to do with the resources offered to them by Mesos.
- Marathon offers a highly available framework delivering:
  - ◇ **Service discovery:** through a dedicated DNS service, as well as other options.
  - ◇ **Load balancing:** through HAProxy.

- ◇ **Constraint management:** to control where in the cluster certain workloads run, maintain a set level of resources for those workloads, enable rack awareness and other constraints.
- ◇ **Applications:** the long-running services you want to run on the cluster; maybe Docker containers but can also be other types of workload.
- ◇ **REST API:** deploy, alter and destroy workloads.

## Notes

### 1.26 Docker Use-Cases

- Container Lifecycles
- Bridging Development Environments

### 1.27 Microservices

- The microservices architectural style is an approach to developing a single application as a suite of small services
- Each service runs in its own process and communicates with lightweight mechanisms, often an HTTP resource API
- These services are built around business capabilities that can be independently and automatically deployed

### 1.28 Microservices and Containers / Clusters

- Containers are excellent for microservices, as they isolate the services.
- Containerization of services makes it easier to manage and update each service
- Docker has led to the emergence of frameworks for managing complex scenarios, such as:
  - ◇ how to manage single services in a cluster
  - ◇ how to manage multiple instances in a service across hosts

- ◇ how to coordinate between multiple services on a deployment and management level
- The Kubernetes container orchestration tool allows easy deployment and management of individual as well as clusters of Docker containers
- With Kubernetes you describe the number of instances, CPU, RAM, etc for the Docker images you need to deploy

## Notes

### 1.29 Summary

In this chapter we covered:

- Infrastructure Terminology
- Virtualization
- Containers
- Container Orchestration
- Docker Use-Cases
- Microservices



## Chapter 2 - Docker in Action

---

### *Objectives*

Key objectives of this chapter

- Docker Basics
- Installation
- Architecture
- Images
- Containers
- Integration
- Services
- Commands
- Volumes
- Creating Dockerfiles
- Compose
- Environment Variables

### 2.1 Docker Basics



- Docker is an open-source (and 100% free) project for IT automation
- You can view Docker as a system or a platform for creating virtual environments which are extremely lightweight virtual machines
- Docker allows developers and system administrators to quickly assemble, test, and deploy applications and their dependencies inside Linux containers supporting the multi-tenancy deployment model on a single

host

- Docker's lightweight containers lend themselves to rapid scaling up and down
  - ◇ *Note:* A container is a group of controlled processes associated with a separate tenant executed in isolation from other tenants
- Written in the Go programming language

### Notes:

The Go programming language (also referred to as *golang*) was developed at Google in 2007 and release in 2009. It is a compiled language – it does not require a VM to run it (like in C# or Java) – with automated garbage collection. Go offers a balance between type safety and dynamic type capabilities; it supports imperative and concurrent programming paradigms.

## 2.2 Where Can I Run Docker?

- Docker runs on any modern-kernel 64-bit Linux distributions
- The minimum supported kernel version is 3.10
  - ◇ Kernels older than 3.10 lack some of the features required by Docker containers
- You can install Docker on VirtualBox and run it on OS X or Windows
- Docker can be installed natively on Windows using Docker Machine, but requires Hyper-V
- Docker can be booted from the small footprint Linux distribution *boot2docker*

## 2.3 Installing Docker Container Engine

- Installing on **Linux**:
  - ◇ Docker is usually available via the package manager of the distributions
  - ◇ For example, on Ubuntu and derivatives:

```
sudo apt-get update && sudo apt install docker.io
```

- Installing on **Mac**

- ◇ Download and install the official Docker.dmg from [docker.com](https://docker.com)
- Installing on **Windows**
  - ◇ Hyper-V must be enabled on Windows
  - ◇ Download the latest installer from [docker.com](https://docker.com)

## 2.4 Docker Toolbox

- **When installing Docker on Mac or PC you are typically installing something called the Docker Toolbox which contains various components including Docker and related tools**
- **The toolbox consists of the following components:**
  - ◇ **Docker Engine** – This is used as the base engine or Docker daemon that is used to run Docker containers.
  - ◇ **Docker Machine** – for running Docker machine commands.
  - ◇ **Docker Compose** for running Docker compose commands.
  - ◇ **Kinematic** – This is the Docker GUI built for Windows and Mac OS.
  - ◇ **Oracle virtualbox**

## 2.5 What is Docker?

- **Docker is a Container runtime.**
- **Docker Container operations include Building, Packaging, Sharing, and Running of Containers**
- **Some common Docker commands include:**
  - ◇ **docker build** - to build a container
  - ◇ **docker start/stop** - to start or stop a container
  - ◇ **docker ps** - lists containers
  - ◇ **docker rm** - remove a container
- **Docker command reference**

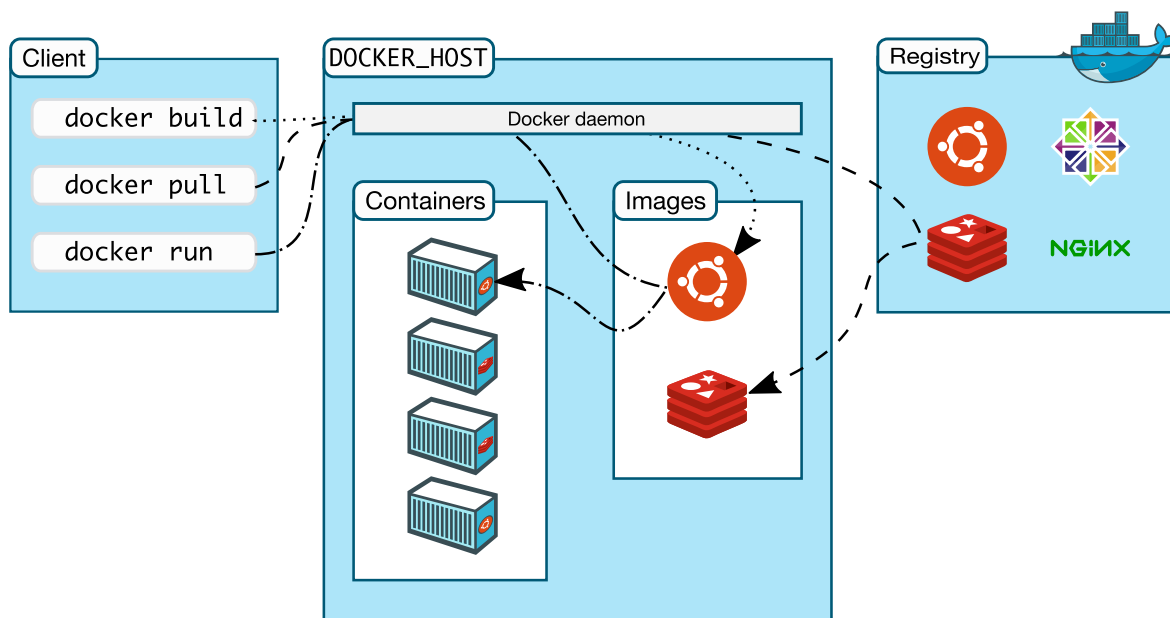
- ◇ <https://docs.docker.com/engine/reference/commandline/docker/>

## 2.6 Docker Architecture

- **Docker consists of these components**
  - ◇ docker daemon
  - ◇ docker REST api
  - ◇ docker CLI Client
- **Using these components docker manages**
  - ◇ containers
  - ◇ images
  - ◇ network settings
  - ◇ data volumes

## 2.7 Docker Architecture Diagram

- **Diagram**



## 2.8 Docker Images

- **Docker Images are read-only templates with instructions for creating Docker containers**
- **Docker Hub includes many standard images such as ones that support various operating systems, databases or programming languages.**
- **Images can be built on top of other images. For example you might create an image for a node application based on the official node image.**
- **Images are based on instructions contained in Dockerfiles**
- **Creating an image:**
  - ◇ create a Dockerfile
  - ◇ add instructions to the file
  - ◇ build the image
- **Running an image**
  - ◇ run the image
  - ◇ this creates a running container based on the image

## 2.9 Docker Containers

- **Containers are created by running Images**
- **Docker has commands for managing containers that let you Create, Run, Manage and Delete containers.**
  - ◇ **docker run** - run a container
  - ◇ **docker rm** - remove a container
- **Containers can be customized by adding various parameters to the run command:**
  - ◇ **--publish** - for forwarding ports
  - ◇ **--detach** - to run the container in the background

- ◇ **--name** - gives the container a name making it easier to manage
- **Run command documentation:**  
<https://docs.docker.com/engine/reference/commandline/run/>

## 2.10 Docker Integration

















- Docker can be integrated with a number of IT automation tools that extend its capabilities, including
  - ◇ Ansible
  - ◇ Chef
  - ◇ Jenkins
  - ◇ Puppet
  - ◇ Salt
- Docker is also deployed on a number of Cloud platforms
  - ◇ Amazon Web Services
  - ◇ Google Cloud Platform
  - ◇ Microsoft Azure
  - ◇ OpenStack
  - ◇ Rackspace

## 2.11 Docker Services

- Docker deployment model is application-centric and in this context provides the following services and tools:
  - ◇ A uniform format for bundling an application along with its dependencies which is portable across different machines
  - ◇ Tools for automatic assembling a container from source code: make, maven, Debian packages, RPMs, etc.
  - ◇ Container versioning with deltas between versions

## 2.12 Docker Application Container Public Repository

- Docker community maintains the repository for official and public domain Docker application images: <https://hub.docker.com>

 <b>postgres</b>  The PostgreSQL object-relational database system provides reliability and data integrity.	2 days ago	 593	 906990
 <b>mongo</b>  MongoDB document databases provide high availability and easy scalability.	21 hours ago	 518	 1076836
 <b>mysql</b>  MySQL is a widely used, open-source relational database management system (RDBMS).	2 days ago	 553	 3368805
 <b>redis</b>  Redis is an open source key-value store that functions as a data structure server.	2 days ago	 599	 2053739

## 2.13 Docker Run Command

- The following commands are shown as executed by the root (privileged) user:

**docker run ubuntu echo 'Yo Docker! '**

- ◇ This command will create a docker container based on the *ubuntu* image, execute the *echo* command on it, and then shuts down

**docker ps -a**

- ◇ This command will list all the containers created by Docker along with their IDs

## 2.14 Starting, Inspecting, and Stopping Docker Containers

**docker start -i <container\_id>**

- ◇ This command will start an existing stopped container in interactive (-i) mode (you will get container's STDIN channel)

**docker inspect <container\_id>**

- ◇ This command will provide JSON-encoded information about the running container identified by *container\_id*

#### **docker stop <container\_id>**

- ◇ This command will stop the running container identified by *container\_id*
- For the Docker command-line reference, visit <https://docs.docker.com/engine/reference/commandline/cli/>

## **2.15 Docker Volume**

- If you destroy a container and recreate it, you will lose data
- Ideally, data should not be stored in containers
- Volumes are mounted file systems available to containers
- Docker volumes are a good way of safely storing data outside a container
- Docker volumes can be shared across multiple containers

- Creating a Docker volume

```
docker volume create my-volume
```

- Mounting a volume

```
docker run -v my-volume:/my-mount-path -it ubuntu:12.04  
/bin/bash
```

- Viewing all volumes

```
docker volume ls
```

- Deleting a volume

```
docker volume rm my-volume
```

## **2.16 Dockerfile**

- Rather than manually creating containers and saving them as custom images, it's better to use Dockerfile to build images
- Sample script

```
# let's use ubuntu docker image  
FROM openjdk
```



```
RUN apt-get update -y
RUN apt-get install sqlite -y
```

```
# deploy the jar file to the container
COPY SimpleGreeting-1.0-SNAPSHOT.jar
/root/SimpleGreeting-1.0-SNAPSHOT.jar
```

- The Dockerfile filename is case sensitive. The 'D' in Dockerfile has to be uppercase.
- Building an image using docker build. (Mind the space and period at the end of the docker build command)

```
docker build -t my-image:v1.0 .
```

- Or, if you want to use a different file name:

```
docker build -t my-image:v1.0 -f mydockerfile.txt
```

## 2.17 Docker Compose

- A container runs a single application. However, most modern application rely on multiple service, such as database, monitoring, logging, messages queues, etc.
- Managing a forest of containers individually is difficult
  - ◇ Especially when it comes to moving the environment from development to test to production, etc.
- Compose is a tool for defining and running multi-container Docker applications on the same host
- A single configuration file, docker-compose.yml, is used to define a group of container that must be managed as a single entity

## 2.18 Using Docker Compose

- Define as many Dockerfile as necessary
- Create a docker-compose.yml file that refers to the individual Dockerfile
- Sample Dockerfile

```
version: '3'
services:
  greeting:
    build: .
    ports:
      - "8080:8080"
    links:
      - mongodb
  mongodb:
    image: mongodb
    environment:
      MONGO_INITDB_ROOT_USERNAME: wasadmin
      MONGO_INITDB_ROOT_PASSWORD: secret
    volumes:
      - my-volume:/data/db
volumes:
  my-volume: {}
```

## 2.19 Dissecting docker-compose.yml

- The Docker Compose file should be named either docker-compose.yml or docker-compose.yaml
  - ◇ Using any other names will require to use the -f argument to specify the filename
- The docker-compose.yml file is writing in YAML
  - ◇ <https://yaml.org/>
- The first line, version, indicates the version of Docker Compose being used
  - ◇ As of this writing, version 3 is the latest

## 2.20 Specifying services

- A 'service' in docker-compose parlance is a container
- Services are specified under the service: node of the configuration file
- You choose the name of a service. The name of the service is meaningful

within the configuration

- A service (container) can be specified in one of two ways: Dockerfile or image name
- Use build: to specify the path to a Dockerfile
- Use image: to specify the name of an image that is accessible to the host

## 2.21 Dependencies between containers

- Some services may need to be brought up before other services
- In docker-compose.yml, it is possible to specify which service relies on which using the links: node
- If service C requires that service A and B be brought up first, add a link as follows:

```
A:
  build: ./servicea
```

```
B:
  build: ./serviceb
```

```
C:
  build: ./servicec
```

```
  link:
```

```
  - A
```

```
  - B
```

- It is possible to specify as many links as necessary
  - ◇ Circular links are not permitted (A links to B and B links to A)

## 2.22 Injecting Environment Variables

- In a microservice, containerized application, environment variables are often used to pass configuration to an application
- It is possible to pass environment variable to a service via the docker-compose.yml file

```
myservice:
  environment:
```

```
MONGO_INITDB_ROOT_USERNAME: wasadmin  
MONGO_INITDB_ROOT_PASSWORD: secret
```

## 2.23 Summary

In this chapter we covered:

- Docker Basics
- Installation
- Architecture
- Images
- Containers
- Integration
- Services
- Commands
- Volumes
- Creating Dockerfiles
- Compose
- Environment Variables

## Chapter 3 - Managing Docker State

---

### *Objectives*

Key objectives of this chapter

- State in Docker
- Volumes
- Bind Mounts
- tmpfs Mounts
- Storing data in the Container
- Storage Drivers
- Networking
- The Default Bridge Network
- User-Defined Bridge Networks

### 3.1 State and Data in Docker

- Docker containers exist as processes in memory. When they are terminated any data that was held in the instance goes away.
- In some cases you have data that is managed by a container and you want that data to stick around after the container terminates.
- The following can be used to manage Local data in Docker:
  - ◇ Volumes - stores data on the host file system, managed by docker
  - ◇ Bind mounts - maps to folders on the host filesystem
  - ◇ tmpfs mounts - for temporary (non-persistent) data
  - ◇ Overlay fs - stores temporary data to the root file system of the container

### Notes

Bind mounts are an older way to save data that are more limited than the volumes which are newer.

## 3.2 Volumes

- Volumes are host OS directories that are mapped to directories in a container.
- Data saved to a volume persists after the container is terminated.
- Deleting the container does not delete the data either.
- It is typical to have volumes that are unnamed
- Naming a volume allows it to be referenced by multiple containers.
- 'volume drivers' allow you to store data remotely

## 3.3 More About Volumes

- Volumes are easier to back up or migrate than bind mounts.
- Volumes can be managed using Docker CLI commands or Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you encrypt the contents of volumes.
- New volumes can have their content pre-populated by a container.
- Using a volume does not increase the size of the container.
- Volumes exist outside of the container's lifecycle

## 3.4 Uses for Volumes

- Saving data without having to specify an existing host directory
- Sharing data between running containers
- Storing data remotely (using drivers, not on the host)
- Migrating data between docker hosts

## Notes

Volumes are created the first time they are mounted into a container. When the container stops the volume still exists. This means that other containers can access the same data by simply accessing the same volume.

When you mount an empty volume into a non-empty container directory the contents of the container directory are copied into the volume. You might want to do this when the contents of a container directory are needed by another container which gains access to it when mounting the same volume.

Volumes do not depend on specific host directories.

## 3.5 Working With Volumes

- Volume Commands
  - ◇ **docker volume** - show docker volume commands
  - ◇ **docker volume create** - Create a volume
  - ◇ **docker volume inspect** - Display detailed information on one or more volumes
  - ◇ **docker volume ls** - List volumes
  - ◇ **docker volume rm** - Remove one or more volumes
  - ◇ **docker volume prune** - Remove all unused volumes

## 3.6 Create Volume

- Create Volume

```
docker volume create myvol1
```

- Inspect Volume

```
docker volume inspect myvol1
```

```
[
  {
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/myvol1/_data",
    "Name": "myvol1",
```

```
"Options": {},  
"Scope": "local"  
}  
]
```

## Notes

The mountpoint shown here can only be modified by docker.

## 3.7 Use Volumes with Containers

Volumes are associated with containers via the docker run command's '-v' parameter

```
-v myvol1://usr/share/nginx/html
```

The above command mounts the 'myvol1' volume to Nginx's html directory inside the container

### ■ Example:

```
docker volume create myvol1  
docker run --name web -v myvol1://usr/share/nginx/html -p 80:80 nginx  
docker cp ./index.html web:/usr/share/nginx/html/index.html  
curl http://localhost
```

### ■ The volume can be accessed from other containers as well

```
docker run -it --name temp -v myvol1:/root alpine sh
```

The above command creates a temporary container with access to the volume and opens a shell for us to interact with it

## 3.8 Bind Mounts

- While 'Volumes' are preferred, bind mounts are also an option in the following situations:
  - ◇ Sharing source code or build artifacts between a dev environment on a Docker host and a container that performs the build.
  - ◇ When the host directory structure is guaranteed to match that of the bind mounts in the container
  - ◇ Sharing existing config files from host machines to containers



## Notes

In the case of Docker being used to run an application in production the Dockerfile would copy the production artifacts into the image rather than using a bind mount.

### 3.9 Using Bind Mounts

- Like volumes Bind mounts are configured using the docker run command's "-v" parameter

**-v /hostdir://usr/share/nginx/html**

Here "/hostdir" is the absolute path to a directory on the host

And the mount point "**//usr/share/nginx/html**" is the dir where nginx serves files from

- The full command to run nginx in a container with a bind mounted home directory

```
docker run --name web -v /hostdir://usr/share/nginx/html:z -p 80:80 nginx
```

- The ":z" on the end of the volume parameter value is required in some cases to avoid permissions errors

## Notes

// full example

```
cd /
mkdir hostdir
cd /hostdir
vi index.html {create file and add some contents}
docker run --name web -v /hostdir://usr/share/nginx/html:z -p 80:80 nginx
curl http://localhost
```

### 3.10 tmpfs Mounts

- tmpfs stands for Temporary File System
- tmpfs mounts are used when you are working with data that does not need

to be persisted. For example:

- ◇ You don't want sensitive data hanging around for security reasons
- ◇ The amount of data is considerable and it is not needed except when the container is running.

- Example

```
--mount type=tmpfs,destination=/app,tmpfs-mode=1770
```

## Notes

'--mount' is a docker run cmd parameter introduced in docker 17.06

Use the run cmd '--tmpfs' parameter instead with earlier versions, like this:

```
--tmpfs /usr/share/nginx/html/
```

## 3.11 Storing Data in the Container

- Containers consist of several read-only layers and one writable layer on top. When data is stored in the container it is stored in this writable layer.
- Commands in a Dockerfile are often used to copy initial data such as source code or dependencies into a container.
- Data created by a container as its running is saved by default in the container.
- Data created and saved in a container when it runs sometimes needs to be saved outside of the container for use later on or by another container. For this purpose the data is often copied to a volume associated with the container.
- The docker 'exec' cmd allows you to access the container's writable layer by opening a shell into the container.

```
docker exec -it mongodb bash
```

The above cmd results in a shell prompt where you can interact with the os and filesystem in the container

## Notes

The docker 'cp' or copy command can also be used to copy files from your host system into the container's file system:

```
docker cp ./file2.html web:/usr/share/nginx/html/file2.html
```

./ = the current host system dir

web = name of container you are copying into

/usr/share... = the directory and file name in the container you are copying into

## 3.12 Storage Drivers

- Docker uses a storage driver to manage data between the read-only layers of a container and the writable top layer.
- Docker uses something called the copy-on-write (CoW) strategy which allows changes to files from the read-only layers of a container by first copying them up to the writable layer.
- The most common storage driver is "overlay2". Other drivers like, devicemapper, btrfs, zfs or vfs may be used depending on the OS or special needs.
- For more information on alternative storage drivers see:  
<https://docs.docker.com/storage/storagedriver/select-storage-driver/>
- The storage driver selection can't be customized in development versions of Docker like Docker Desktop for Mac or Desktop for Windows.
- Information about the current storage driver is available through the "docker info" command.

## 3.13 Remote Data Storage

- In some cases you need to:
  - ◇ Store data persistently
  - ◇ Have the data be accessible over the network
- For these requirements you can use a remote volume
- Remote volumes are accessible through Docker Volume Plugins like

these:

- ◇ **REX-ray** plugin - Provides access to VirtualBox, EC2, Google Compute Engine, OpenStack, and EMC volumes
- ◇ **S3 Storage plugin** - Provides access to Microsoft Azure file storage shares over SMB
- More information can be found on these plugins here:
  - ◇ <https://github.com/rexray/rexray>
  - ◇ <https://github.com/Azure/azurefile-dockervolumedriver>

## Notes

More information on Docker Volume Plugins can be found here:

[docs.docker.com/engine/extend/legacy\\_plugins/](https://docs.docker.com/engine/extend/legacy_plugins/)

## 3.14 Networking

- Docker containers are commonly used to run the various components of a networked application.
- Networks can be configured in Docker using the following drivers:
  - ◇ **Bridge** - with the default 'bridge' driver a network defined by the user allows multiple containers on the same Docker host to communicate
  - ◇ **Host** - uses the Docker host's network while isolating other aspects of the container.
  - ◇ **Overlay** - for when containers running on different Docker hosts (on different physical machines) need to communicate.
  - ◇ **Macvlan** - for when you need your containers to look like physical hosts, each with its own unique MAC address.
  - ◇ With **third-party network plugins** you can integrate Docker with specialized network stacks.

## Notes

### 3.15 The Default Bridge Network

- Only applies to containers running on the same Docker host
- The default network named 'bridge' is created automatically when you start Docker.
- All containers that don't specify a network (with `--network`) use the default bridge network.
- When using the default network:
  - ◇ Containers must address each other using IP addresses
  - ◇ All containers use the same settings
  - ◇ Configuring the default network requires a Docker restart
  - ◇ Removing a container from the default network requires a restart.

## Notes

User-defined bridge networks are considered to be better than the default especially for production situations.

### 3.16 User-Defined Bridge Networks

- Are created using the '**docker network create**' command
- Are removed using the '**docker network rm**' command
  - ◇ Any connected containers should be disconnected first
- Provide automatic DNS resolution between containers
- Provide better isolation
- Only containers specifically connected to the network are attached.
- Containers can be connected/disconnected from user-defined bridge networks on the fly.

## Notes

```
// connect my-nginx container to the my-net network
docker network connect my-net my-nginx
```

## 3.17 Docker Network Commands

### ■ Commands

```
docker network - Show available commands
docker network create - Create a network
docker network connect - Connect a container to a docker network
docker network disconnect - Disconnect a container from a network
docker network inspect - Display detailed network information
docker network ls - List networks
docker network rm - Remove one or more networks
docker network prune - Remove all unused networks
```

## 3.18 Creating a User-Defined Bridge Network

### ■ Create a Network named 'mynet'

```
docker network create mynet
```

### ■ Run a server that uses the 'mynet' network

```
docker run -it -d --name web --network mynet nginx
```

### ■ Create a client container that uses the same network

```
docker run -it --network mynet --name cl alpine
```

### ■ Add the CURL command and make a network call from the client to the server across 'mynet'

```
apk add curl
curl http://web
```

### ■ Note how the name of the server container "web" is when referencing the server from the client

## Notes

## 3.19 Summary

In this chapter we covered:

- State in Docker
- Volumes
- Bind Mounts
- tmpfs Mounts
- Storing data in the Container
- Storage Drivers
- Networking
- The Default Bridge Network
- User-Defined Bridge Networks





## Chapter 4 - Open Container Initiative and Container Runtime Interface

---

### *Objectives*

Key objectives of this chapter

- OCI
- Docker architecture
- CRI-O

### 4.1 Open Container Initiative (OCI)

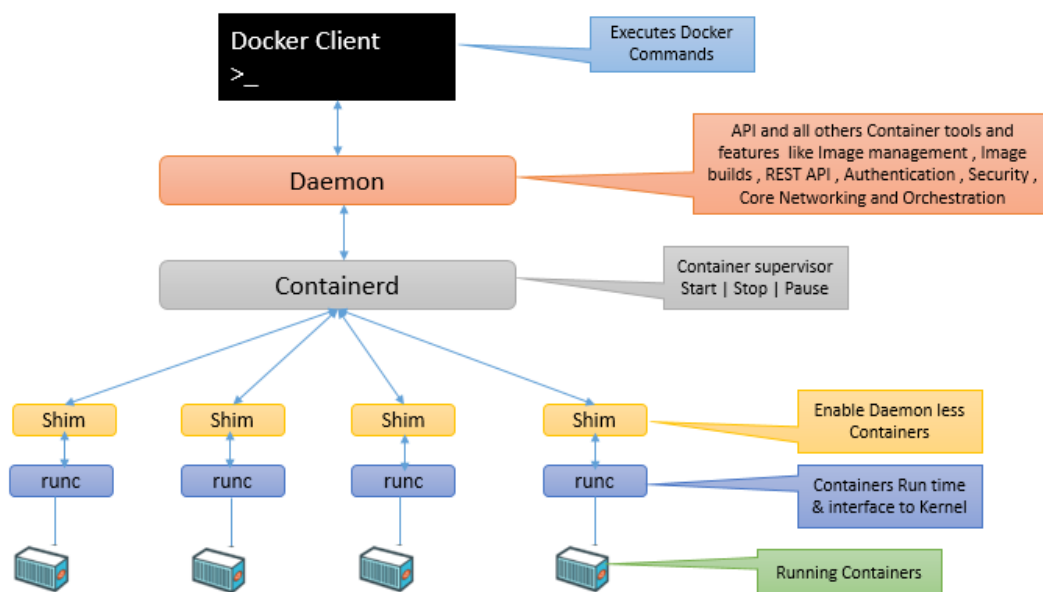
- The Open Container Initiative (OCI) is a lightweight and open governance structure (project) formed by Docker, CoreOS, and other leaders in the container industry.
- It was formed for the purpose of creating open industry standards around container formats and runtime.
- The OCI currently contains two specifications:
  - ◇ Image Specification (image-spec).
  - ◇ Runtime Specification (runtime-spec)
- At a high-level an OCI implementation would download an OCI Image then unpack that image into an OCI Runtime filesystem bundle. At this point the OCI Runtime Bundle would be run by an OCI Runtime.

### 4.2 Docker

- Before version 1.11, Docker's implementation was a monolithic daemon and the name **Docker** itself became synonymous with the word **container**.
- The centralized Docker service did everything as one package, such as downloading container images, launching container processes, exposing a service, and acting as a log collection daemon.
- The monolithic architecture does not follow best practices for the Unix process and privilege separation.

- This led to the splitting of Docker into different components when 1.11 was launched.
  - ◇ “We are excited to introduce Docker Engine 1.11, our first release built on runC™ and containerd™. With this release, Docker is the first to ship a runtime based on OCI technology, demonstrating the progress the team has made since donating our industry-standard container format and runtime under the Linux Foundation in June of 2015” (Source Docker).

### 4.3 Docker Engine Architecture



Source: Adapted from <https://www.dclessons.com/how-docker-engine-works>

### 4.4 runC

- runC is a lightweight container runtime.
- Docker donated its container format and runtime, runC, to the OCI to serve as the cornerstone of this new effort.
  - ◇ It is available now at <https://github.com/opencontainers/runc>.
- It was originally a low-level Docker component.

- It has since been rolled out as a standalone modular tool.
- Its purpose is to improve the portability of containers by providing a standardized interoperable container runtime that can work both as part of Docker and independently of Docker in alternative container systems.
- runC can help you avoid being strongly tied to specific technologies, hardware, or cloud service providers.
- runC is the most widely used container runtime.
  - ◇ There are others such as crun, railcar, and katacontainers

## **4.5 containerd**

- Like runC, containerd is another core building block of the Docker system that has been separated off as an independent open-source project.
- containerd is a daemon that acts as an interface between the container engine and container runtimes.
- It is supported by both Linux and Windows
- It provides an abstracted layer, API, that makes it easier to manage container lifecycles, such as image transfer, container execution, snapshot functionality, and certain storage operations, using simple API requests.
- It makes containers more portable since you don't have to rely on platform-specific, low-level system calls to manage container lifecycle

## **4.6 containerd Benefits**

- You get the following benefits without delving into the underlying OS details:
  - ◇ Push and pull functionality
  - ◇ Image management APIs to create, execute, and manage containers and their tasks
  - ◇ Snapshot management.

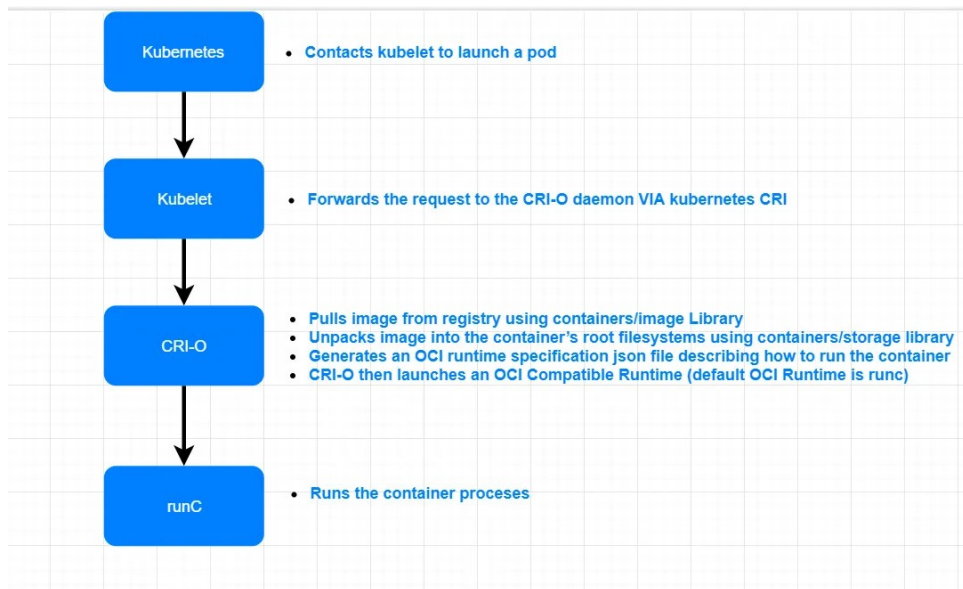
## 4.7 CRI-O

- Before CRI-O, Kubernetes was bound to a specific container runtime which spawned a lot of maintenance overhead for the Kubernetes community.
- Container Runtime Interface (CRI) is a plugin-based interface that gives kubelet the ability to use different OCI-compliant container runtimes, such as runC.
- With the plugin/interface approach, you can use a different runtime without recompiling Kubernetes.
- CRI-O makes decouples various runtimes from the Kubernetes container runtime.

## 4.8 CRI-O Components

- **OCI compatible runtime** – Default is runC, other OCI compliant are supported as well e.g Kata Containers.
- **containers/storage** – Library used for managing layers and creating root filesystems for the containers in a pod.
- **containers/image** – Library is used for pulling images from registries.
- **networking (CNI)** – Used for setting up networking for the pods.
- **container monitoring (conmon)** – Utility within CRI-O that is used to monitor the containers.

## 4.9 Kubernetes and CRI-O



## 4.10 Using Container Runtimes with Minikube

- You can specify a container runtime that Kubernetes should use when launching a cluster with minikube.

```
minikube --container-runtime=docker (deprecated)
minikube --container-runtime=containerd
minikube --container-runtime=crio
```

- Checking what runtime is used by Kubernetes

```
kubectl get nodes -o wide
```

- Sample output for CONTAINER-RUNTIME column:

```
cri-o://{VERSION}
docker://{VERSION}
containerd://{VERSION}
```

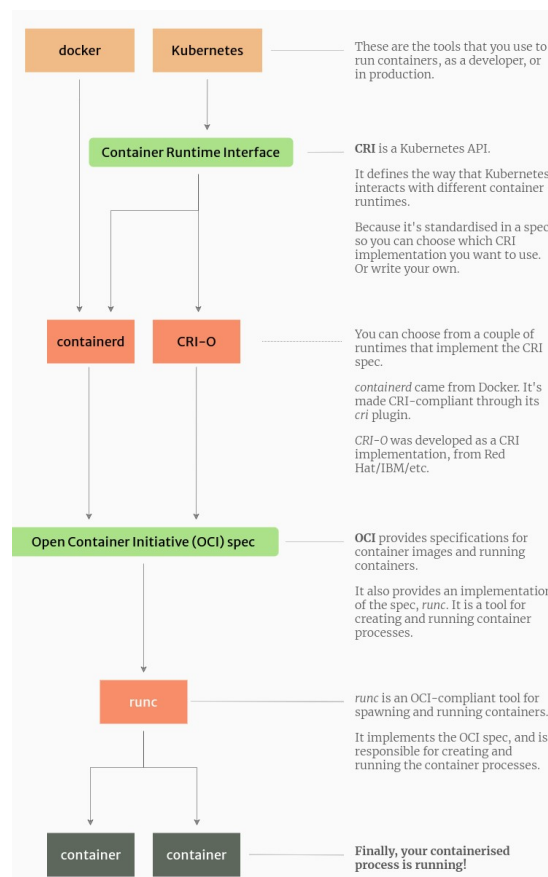
## 4.11 Docker Runtime and Kubernetes

- The Docker runtime is deprecated in Kubernetes v1.20+ and will be completely removed by late 2021.

- Moving forward, Docker will be used to create images using the same Dockerfile scripts that you are aware of. Just the Docker container runtime support is deprecated in Kubernetes.
- Due to containerd and runC components in Docker, the Docker images are OCI-compliant images and can be utilized by other runtimes, such as CRI-O or containerd
  - ◇ AKS, EKS, GKE defaults to containerd
- As developers, nothing would change.
- As administrators switch from the Docker runtime to some other CRI-based runtime, such as CRI-O.

### 4.12 Putting Things Together

- Docker, Kubernetes, OCI, CRI-O, containerd, and runc



source: <https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/>

## 4.13 Summary

- In this chapter, you learned the following:
  - ◇ OCI
  - ◇ Docker architecture
  - ◇ containerd
  - ◇ runC
  - ◇ CRI-O





## Chapter 5 - Kubernetes Architecture

---

### *Objectives*

Key objectives of this chapter

- Architecture Diagram
- Components
- Cluster
- Master
- Node
- Pod
- Container
- Interaction through API

### 5.1 Kubernetes Basics

- Kubernetes is Greek for "helmsman" or "pilot"
- Originally founded by Joe Beda, Brendan Burns and Craig McLuckie
- Kubernetes is commonly referred to as **K8s**
- An open-source system for automating deployment, scaling, and management of containerized applications
- Most often associated with Docker but supports other container types as well

### Notes

### 5.2 What is Kubernetes?

Kubernetes:

- Is an open-source system for automating deployment, scaling, and management of containerized applications
- Groups application containers into logical units for easy management and

discovery.

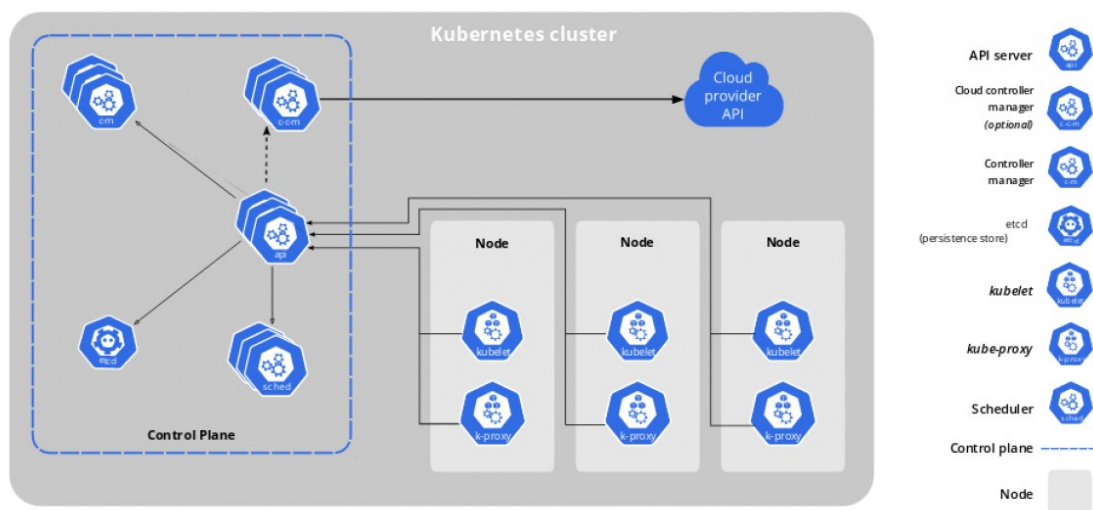
- Is designed to easily scale applications
- Can be implemented on-premises or in hybrid, or public cloud infrastructures
- Can be deployed on a bare-metal cluster (real machines) or on a cluster of virtual machines.

### 5.3 Container Orchestration

- Container orchestration is the primary responsibility of Kubernetes
- Containers involved in workflows are scheduled to run on physical or virtual machines
- The health of running containers is monitored so that dead, unresponsive, or otherwise healthy containers can be replaced.
- Supports application and container clustering
- Is particularly useful in implementing microservices

### 5.4 Architecture Diagram

- This chapter will review various parts of the following architecture diagram:



source: <https://kubernetes.io/>

## Notes

### 5.5 Components

- **Cluster** - Includes one or more master and worker nodes
- **Master** - Manages nodes and pods
- (worker) **Node** - a physical, virtual or cloud machine
- **Pod** - A group of one or more containers, created and managed by kubernetes
- **Container** - Are most commonly Docker containers where application processes are run
- **Volume** - A directory of data accessible to containers in a pod. It shares lifetime with the pod it works with.
- **Namespace** - A virtual cluster. Allows for multiple virtual clusters within a physical one.
- **Label** - assigned to a worker node so you can run a specific type of workloads (or application) on a designated worker node.
- **Annotation** - non-identifying metadata associated to objects.
- **Controllers** - responsible for the configuration and health of the cluster's components

## Notes

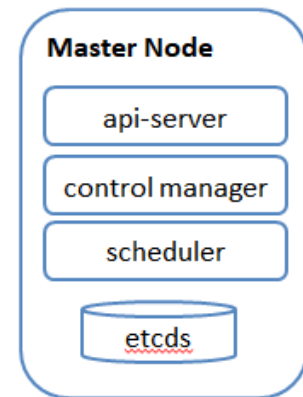
### 5.6 Kubernetes Cluster

- A Kubernetes cluster is a set of machines(nodes) used to run containerized applications.
- To do work a clusters need to have at least a one master node and one worker node.

- The Master node determines where and what is run on the cluster.
- Worker nodes contain pods which contain containers. Containers hold execution environments where work can be done.
- A cluster is configured via the kubectl command line interface or by the Kubernetes API

## 5.7 Master Node

- The Master node manages worker nodes.
- The master node includes several components:
  - ◇ Kube-APIServer - traffic enters the cluster here
  - ◇ Kube-Controller-Manager - runs the cluster's controllers
  - ◇ Etcd - Maintains cluster state, provides key-value persistence
  - ◇ Kube Scheduler - schedules activities to worker nodes
- Clusters can have **more than one** master node
- Clusters can have **only one active** master node



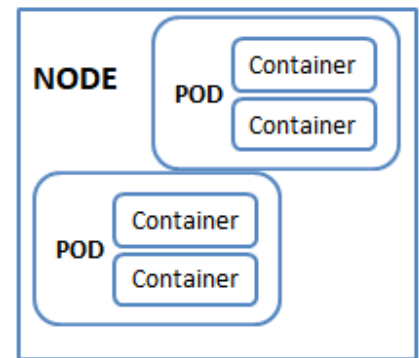
## 5.8 Kube-Control-Manager

- The Kube-Control-Manager (part of the Master Node) manages the following controllers:
  - ◇ Node controller
  - ◇ Replication controller
  - ◇ Endpoints controller
  - ◇ Service account controller

- ◇ Token controller
- All these controller operations are compiled into a single application.
- The controllers are responsible for the configuration and health of the cluster's components

## 5.9 Nodes

- A node consists of a physical, virtual or cloud machine where Kubernetes can run Pods that house containers.
- Clusters have one or more nodes
- Nodes can be configured manually through kubectl
- Nodes can also self-configure by sending their information to the Master when they start up
- Information about running nodes can be viewed with kubectl



## Notes

Other components found on the worker node include:

- kubelet - interacts with master node, manages containers and pods on the node
- kube-proxy - responsible for network configuration
- container runtime - responsible for running containers in the pods (typically Docker)

## 5.10 Pod

- A pod is the unit of work on Kubernetes
- Pods provide a way to group one or more containers

- Pods provide a solution for managing containers that depend on each other and need to cooperate on the same host
- Pods are considered throwaway entities that can be discarded and replaced as needed
- Each pod gets a unique ID (UID)
- Pods are always scheduled together and always run on the same machine
- All the containers in a pod have the same IP address and port space
- The containers within a pod can communicate using localhost or standard inter-process communication
- Containers within a pod have access to the shared local storage on the node hosting the pod

### 5.11 Using Pods to Group Containers

- The benefits of grouping related containers within a pod, as opposed to having one container with multiple applications, are:
  - ◇ **Transparency** – making the containers within the pod visible to the infrastructure enables the infrastructure to provide services to those containers, such as process management and resource monitoring
  - ◇ **Decoupling** software dependencies – the individual containers maybe be versioned, rebuilt, and redeployed independently
  - ◇ **Ease of use** – users don't need to run their own process managers
  - ◇ **Efficiency** – because the infrastructure takes on more responsibility, containers can be more lightweight

### 5.12 Label

- Labels are key-value pairs that are used to group together sets of objects, often pods.
- Labels are important for several other concepts, such as replication controller, replica sets, and services that need to identify the members of the group

- Each pod can have multiple labels, and each label may be assigned to different pods.
- Each label on a pod must have a unique key

### 5.13 Label Syntax

- The label key must adhere to a strict syntax
  - ◇ Label has two parts: prefix and name
  - ◇ Prefix is optional. If it exists then it is separated from the name by a forward slash (/) and it must be a valid DNS sub-domain. The prefix must be 253 characters long at most
  - ◇ Name is mandatory and must be 63 characters long at most. Name must begin with an alphanumeric character and contain only alphanumeric characters, dots, dashes, and underscores. You can create another object with the same name as the deleted object, but the UIDs must be unique across the lifetime of the cluster. UIDs are generated by Kubernetes
  - ◇ Values follow the same restrictions as names

### 5.14 Label Selector

- Label selectors are used to group objects based on their labels
- A value can be assigned to a key name using equality-based selectors, (=, ==, !=).
  - ◇ e.g.
    - role = webserver
    - role = dbserver, application != sales
- **in** operator can be used as a set-based selector
  - ◇ .e.g
    - role in (dbserver, backend, webserver)

## 5.15 Annotation

- Unlike labels, annotation can be used to associate arbitrary metadata with Kubernetes objects.
- Kubernetes stores the annotations and makes their metadata available
- Unlike labels, annotations don't have strict restrictions about allowed characters and size limits

## 5.16 Persistent Storage

- When a pod is destroyed, the data used by the pod is also destroyed.
- If you want the data to outlive the pod or share data between pods, volume concept can be utilized.
- Persistent storage is managed using options, such as:
  - ◇ Persistent Volumes and Claims (PVC)
  - ◇ Secrets
  - ◇ ConfigMaps

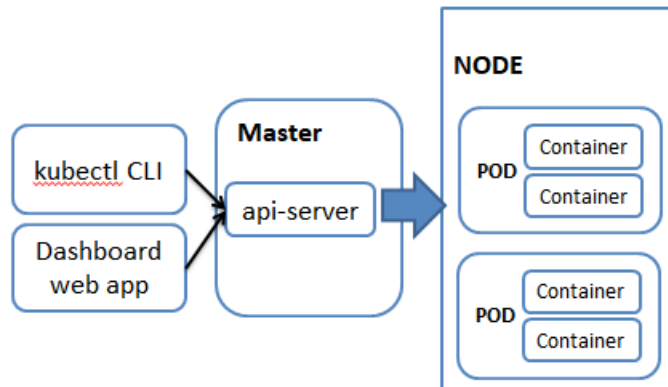
## 5.17 Resource Quota

- Kubernetes allows management of different types of quota
- Compute resource quota
  - ◇ Compute resources are CPU and memory
  - ◇ You can specify a limit or request a certain amount
  - ◇ Uses fields, such as requests.cpu, requests. memory
- Storage resource quota
  - ◇ You can specify the amount of storage and the number of persistent volumes
  - ◇ Uses fields, such as requests.storage, persistentvolumeclaims
- Object count quota



- ◇ You can control API objects, such as replication controllers, pods, services, and secrets
- ◇ You can not limit API objects, such as replica sets and namespaces.

## 5.18 Interacting with Kubernetes



- All user interaction goes through the master node's api-server
- kubectl provides a command line interface to the API
- Control of Kubernetes can also be done through the Kubernetes Dashboard (web UI)

### Notes

## 5.19 Summary

In this chapter we covered:

- Architecture Diagram
- Components
- Cluster
- Master
- Node

- Pod
- Container
- Interaction through API

## Chapter 6 - Working with Kubernetes

---

### *Objectives*

Key objectives of this chapter

- Installation
- Startup
- Kubernetes API tools
- Deployments
- Networks
- Services
- Namespaces
- Useful Commands
- Scheduling
- Node Management

### 6.1 Installation

- Kubernetes is compatible with various environments.
- Before starting out you should know the following information about your intended install:
  - ◇ OS (Linux/Win/Mac)
  - ◇ Intended Usage (Learning or Production Environment)
- Knowing the above information then allows you to choose the proper version, installer and installation procedure from the official Kubernetes Getting Started Site:
  - ◇ <https://kubernetes.io/docs/setup/>
- Minikube provides an easy way to install single-node cluster on a local machine for development or learning
- Minikube's installation procedure creates a default k8s cluster

## Notes

The chapters and labs in this course use the Minikube variation of Kubernetes

### 6.2 Startup

- These commands are helpful when starting up Minikube:
- Docker should be stopped before starting minikube
  - ◇ Stop docker: `sudo systemctl stop docker`
  - ◇ Check docker status: `sudo docker info`
- Minikube Commands
  - ◇ `minikube status`
  - ◇ `minikube start` (add `-p {name}` to create a new cluster)
  - ◇ `minikube stop`
  - ◇ `minikube delete` (deletes the existing cluster)
  - ◇ `minikube ip` (gets the cluster's ip address)

## Notes

### 6.3 Kubernetes Tools

- Kubernetes can be configured/controlled via the following tools:
  - ◇ kubectl Command Line Interface (CLI)
  - ◇ API Proxy (REST interface)
  - ◇ Dashboard Web Application

### 6.4 kubectl Command Line Interface

- kubectl allows you to configure and control each component of the

## Kubernetes cluster

- Some commands include:

Command	Description
<code>kubectl help</code>	Get help on usage
<code>kubectl cluster-info</code>	Shows ip address of the cluster
<code>kubectl create</code>	Create various k8s assets
<code>kubectl describe node {name}</code>	Show configuration of a specific node
<code>kubectl run {image}</code>	Run a docker image
<code>kubectl get nodes</code>	Show a list of the cluster's nodes

- We will be using `kubectl` commands throughout this chapter

## 6.5 API Proxy

- A proxy can be used to access the Kubernetes API over http
- The following cmd runs proxy in its own terminal

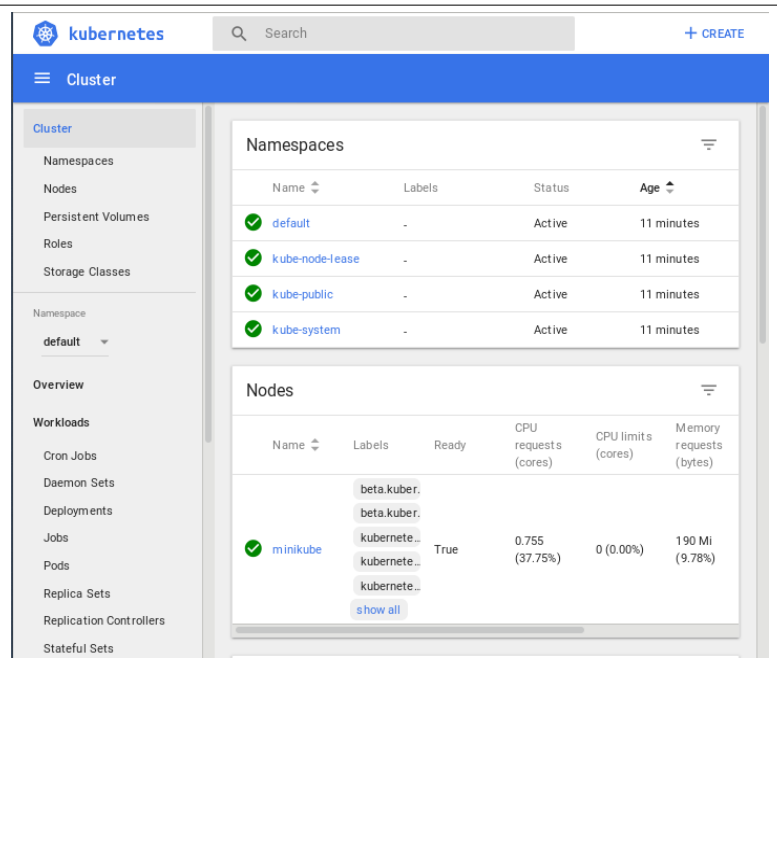
```
kubectl proxy
```

- The proxy can then be accessed via HTTP (browser or curl):

```
curl http://localhost:8001/  
curl http://localhost:8001/api/v1/services/kubernetes  
curl http://localhost:8001/api/v1/namespaces/default/pods
```

## 6.6 Dashboard

- The Dashboard web app is opened using the command:  
`minikube dashboard`
- Dashboard provides GUI access to manage various components of the cluster and to workloads
- From here you can browse and drill down and check the configuration of each component
- The "+Create" button in the upper right corner allows you to create Kubernetes assets or apps



## 6.7 Kubernetes Component Hierarchy

To understand what Kubernetes tools do it helps to remember this hierarchy:

- Kubernetes installations exist as clusters,
- Clusters contain Nodes,
- Nodes run Pods
- Pods hold containers
- Containers run applications

## 6.8 Deployments

- Deployments are the recommended way to manage the creation and scaling of applications. Once a deployment is created Kubernetes:

- ◊ Determines which node will run the app,
- ◊ Creates Pods as needed to hold the app,
- ◊ Determines if the app needs to run on multiple nodes,
- ◊ Schedules the app to be run,
- ◊ Restarts any instances that fail
- Kubernetes creates all components required to run the deployment
- When deployments are deleted Kubernetes deletes all the associated pods and containers as well

## 6.9 Deployment Commands

- Create a basic deployment

```
kubectl create deployment nginx-deployment --image=nginx:latest
```

- Create a deployment using a deployment yaml file

```
kubectl create -f nginx-deployment.yaml
```

- List deployments

```
kubectl get deployments
```

- Check rollout status of a deployment

```
kubectl rollout status deployment nginx-deployment
```

- Check deployment details

```
kubectl get deployment nginx-deployment -o yaml
```

- Delete a deployment

```
kubectl delete deployment nginx-deployment
```

### Notes

-o yaml - tells kubectl you want the output in yaml format (json format is also available)

// an example deployment yaml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

### 6.10 Updating Deployments

- To update a deployment you would:

- ◇ Edit and update its deployment yaml file
- ◇ Run the apply command:

```
kubectl apply -f nginx-deployment.yaml
```

- You can check the details of a running deployment with:

```
kubectl describe deployment nginx-deployment
```

- Recent updates can be rolled back using the **undo** command:

```
kubectl rollout undo deployment.v1.apps/nginx-deployment
```

- Image updates can be rolled back using this command:

```
kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.91
```

- Changes to the replica count are done with the **scale** command:

```
kubectl scale deployment.v1.apps/nginx-deployment --replicas=1
```



## Notes

Two other useful commands are listed here. They allow you to pause a deployment and resume it after making multiple updates. All the updates will then be processed at once.

```
kubectl rollout pause deployment nginx-deployment
...execute update statements
kubectl rollout resume deployment nginx-deployment
```

## 6.11 Network Considerations

- Pods running in K8s run by default on a private, isolated network.
- By default Pods are only visible from other pods within the cluster
- This allows applications in one pod to talk to applications in other pods
- But, pods are NOT initially visible outside the cluster
- This means we can't call applications in pods from outside
- Adding a K8s **Service** makes it possible to call applications from outside

## 6.12 Services

- Services expose an application running on a set of Pods as a network service that is accessible outside the cluster
- Services are created with the 'expose' command:

```
kubectl expose deployment/server-nginx --type="LoadBalancer" --port 80
```

types include: LoadBalancer, NodePort, ExternalName, ClusterIP

- The command creates a service named 'server-nginx' that exposes the Pod's port 80 to the outside network as a randomly assigned port #.

```
kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP
server-nginx	LoadBalancer	10.111.231.216	<pending>	80:31521/TCP

- We can now access the nginx server using the cluster ip address and the randomly assigned port #31521 :

```
kubectl cluster-info (to get cluster ip address)
curl http://192.168.99.100:31521
```

- Pop up a browser on a service's URL (server:port)

```
minikube service server-nginx
```

## Notes

For more information on the 'expose' command execute the following:

```
kubectl expose --help
```

## 6.13 Namespaces

- Large numbers of assets(deployments, pods, services, etc.) can be hard to manage. Namespaces make it easier by allowing you to organize them.
- Existing namespaces can be listed like this:

```
kubectl get namespaces
```

NAME	STATUS	AGE
default	Active	20h
kube-node-lease	Active	20h
kube-public	Active	20h
kube-system	Active	20h

- Assets are initially created in the '**default**' namespace
- New namespaces are made with the create command:

```
kubectl create namespace myspace
```

- Namespaces can be assigned in an asset's yaml file
- Namespaces can be also assigned with the apply command:

```
kubectl apply --namespace myspace -f assetdef.yml
```

- Lists pf assets obtained with the get command can be narrowed down with the **--namespace** parameter

```
kubectl get pods --namespace=default
```

This only shows pods in the given namespace

## Notes

More information on namespace can be found here:

<https://codingbee.net/tutorials/kubernetes/namespaces>

## 6.14 Labels

- Labels can be assigned to a worker node so you can run a specific type of workloads (or application) on a designated worker node.
- For example, production workloads should be running on specific worker nodes and shouldn't get mixed with staging or development workloads.
- Read node details along with their labels:

```
kubectl get nodes --show-labels
```

- Read a specific node details

```
kubectl label --list nodes node_name
```

- Assign a label to a node

```
kubectl label nodes node1 workload=prod
```

- Override the node label

```
kubectl label --overwrite nodes node1 workload=staging
```

- Remove the node label by providing the key without any value

```
kubectl label --overwrite nodes node1 workload-
```

## 6.15 Annotations

- Kubernetes annotations are key-value pairs.
- Annotations are used to associate arbitrary metadata to objects.
- The metadata in an annotation can be small or large, structured or unstructured, and can include characters not permitted by labels.
- In the following example, the text in bold represents the annotation

```
apiVersion: v1
kind: Pod
metadata:
  name: annotations-example
  annotations:
    imageregistry: "https://hub.docker.com/"
spec:
  containers:
    - name: ubuntu
      image: ubuntu:20.10
```

## 6.16 Other Useful Commands

- Get list of pods

```
kubectl get pods # shows pod names & info
```

- Run container shell commands

```
kubectl exec -it {pod-name} {cmd}  
kubectl exec -it nginx-web-7564864859-zq6jp /bin/bash # opens bash shell
```

- Get logs for a Pod

```
kubectl logs {podname}  
kubectl logs nginx-web-7564864859-zq6jp
```

- Copy files between local file system and container fs

```
kubectl cp {file-name} {pod-name}:{container-fs-dir}  
kubectl cp file2.html nginx-web-7564864859-zq6jp:/tmp
```

## Notes

## 6.17 Summary

In this chapter we covered:

- Installation
- Startup
- Kubernetes API tools
- Deployments
- Networks
- Services
- Namespaces
- Useful Commands
- Scheduling
- Node Management





## Chapter 7 - Kubernetes Workload

---

### *Objectives*

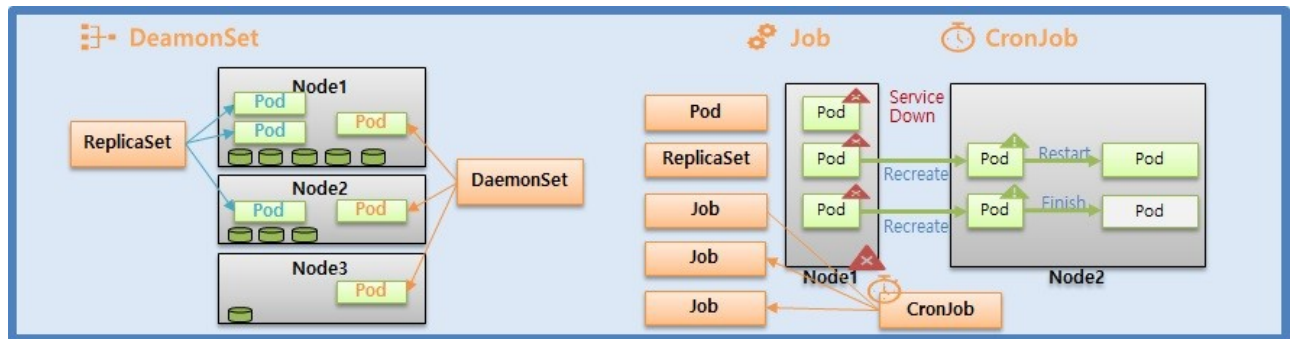
Key objectives of this chapter

- Managing Workload
- Working with Deployments
- Working with StatefulSet
- Working with Jobs
- Working with DaemonSet

### 7.1 Kubernetes Workload

- Workloads are objects you use to manage and run your containers on the cluster
- To deploy and manage your containerized applications and other workloads on your Kubernetes cluster, you create Kubernetes controller objects.
- The controller objects represent the applications, daemons, and batch jobs running on your clusters.
- Kubernetes provides different kinds of controller objects that correspond to different kinds of workloads you can run.
- Certain controller objects are better suited to representing specific types of workloads.
- Some common types of workloads and the Kubernetes controller objects you can create include:
  - ◇ Stateless applications
  - ◇ Stateful applications
  - ◇ Jobs
  - ◇ DaemonSets

## 7.2 Kubernetes Workload (contd.)



source: <https://kubetm.github.io/img/practice/beginner/Controller%20with%20DatemonSet,%20Job,%20CronJob%20for%20Kubernetes.jpg>

## 7.3 Managing Workloads

- You can create, manage, and delete objects using imperative and declarative methods.
- There are 3 ways to manage workloads:
  - ◇ imperative commands
  - ◇ imperative object configuration
  - ◇ declarative object configuration

## 7.4 Imperative commands

- Imperative commands allow you to quickly create, view, update and delete objects with kubectl.
- These commands are useful for one-off tasks or for making changes to active objects in a cluster.
- Imperative commands are commonly used to operate on live, deployed objects on your cluster.
- Imperative commands do not require a strong understanding of object schema and do not require configuration files.
- kubectl features several verb-driven commands for creating and editing Kubernetes objects. For example:



- ◇ **run**: Generate a new object in the cluster. Unless otherwise specified, run creates a Deployment object. run also supports several other generators.
- ◇ **expose**: Create a new Service object to load balance traffic across a set of labeled Pods.
- ◇ **autoscale**: Create a new Autoscaler object to automatically horizontally scale a controller object, such as a Deployment.

## 7.5 Imperative Object Configuration

- Imperative object configuration creates, updates, and deletes objects using configuration files containing fully-defined object definitions.
- You can store object configuration files in source control systems and audit changes more easily than with imperative commands.
- You can run kubectl apply, delete, and replace operations with configuration files or directories containing configuration files.
- Configuration files can be in YAML or JSON format. YAML is most commonly used.
- The details are available on the official website:  
<https://kubernetes.io/docs/reference/generated/kubernetes-api/>

## 7.6 Declarative Object Configuration

- Declarative object configuration operates on locally-stored configuration files but does not require an explicit definition of the operations to be executed.
- Instead, operations are automatically detected per-object by kubectl.
- This is useful if you are working with a directory of configuration files with many different operations.
- Declarative object management requires a strong understanding of object schemas and configuration files.
- You can run kubectl apply to create and updates objects declaratively.

apply updates objects by reading the whole live object, calculating the differences, then merging those differences by sending patch requests to the Kubernetes API server

## 7.7 Configuration File Schema

- A configuration file can involve several attributes. Here's a list of some of the commonly used attributes:
  - ◇ **apiVersion**: api version for the type of the object you are creating. It can be any of the following: v1, v1beta1, v1beta2, .... Refer to the official website to see the possible versions.
  - ◇ **kind**: the type of object you are creating, such as deployment, pod, service, job, secret, replication controller, etc.
  - ◇ **metadata**: information about the "kind". Often "name" is defined that must be unique.
  - ◇ **spec**: actual object specification is defined here.
  - ◇ **replicas**: number of replicas that will be created
  - ◇ **template**: template to be used by all the pods. e.g. label can be assigned to the pods.
  - ◇ ...

## 7.8 Understanding API Version

- Each Kubernetes release uses a different apiVersion.
- **alpha** - early candidate for the new release. These may contain bugs and may not be available in the final release.
- **beta** - the beta features will eventually be included in the new release but the way objects are defined and used may change in the final release.
- **stable** - these are safe to use and do not contain 'alpha' or 'beta' in the name.
- **v1** - the first stable release of the Kubernetes API and contains most core objects

- **apps/v1** - the most common API group in Kubernetes, with many core objects being drawn from it and v1. It includes objects, such as Deployments, RollingUpdates, and ReplicaSets.
- **autoscaling/v1** - allows pods to be autoscaled based on resource utilization

## 7.9 Understanding API Version

- **batch/v1** - contains objects related to batch processing and job-like tasks
- **batch/v1beta1** - a beta release of new functionality for batch objects in Kubernetes. It also includes CronJobs that let you run Jobs at a specific time or periodicity.
- **certificates.k8s.io/v1beta1** - validate network certificates for secure communication in your cluster
- **extensions/v1beta1** - this group is deprecated. Before Kubernetes 1.6, it included many commonly used features of Kubernetes, such as Deployments, DaemonSets, ReplicaSets, and Ingress. These were relocated from extensions to specific API groups (e.g. apps/v1).
- **policy/v1beta1** - adds the ability to set a pod disruption budget and new rules around pod security.
- **rbac.authorization.k8s.io/v1** - includes extra functionality for Kubernetes role-based access control.

## 7.10 Obtaining API Versions

- Depending on your Kubernetes release, API versions would vary.
- Obtaining Kubernetes objects and API versions:
- `kubectl api-resources`
- Here is the sample output for Kubernetes 1.19

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND	VERBS
bindings		v1	true	Binding	[create]
componentstatuses	cs	v1	false	ComponentStatus	[get list]
configmaps	cm	v1	true	ConfigMap	[create delete deletecollection get list patch update watch]
endpoints	ep	v1	true	Endpoints	[create delete deletecollection get list patch update watch]
events	ev	v1	true	Event	[create delete deletecollection get list patch update watch]
lintranges	limits	v1	true	LimitRange	[create delete deletecollection get list patch update watch]
namespaces	ns	v1	false	Namespace	[create delete get list patch update watch]
nodes	no	v1	false	Node	[create delete deletecollection get list patch update watch]
persistentvolumeclaims	pvc	v1	true	PersistentVolumeClaim	[create delete deletecollection get list patch update watch]
persistentvolumes	pv	v1	false	PersistentVolume	[create delete deletecollection get list patch update watch]
Pods	po	v1	true	Pod	[create delete deletecollection get list patch update watch]
podtemplates		v1	true	PodTemplate	[create delete deletecollection get list patch update watch]
replicationcontrollers	rc	v1	true	ReplicationController	[create delete deletecollection get list patch update watch]
resourcequotas	quota	v1	true	ResourceQuota	[create delete deletecollection get list patch update watch]
secrets		v1	true	Secret	[create delete deletecollection get list patch update watch]
serviceaccounts	sa	v1	true	ServiceAccount	[create delete deletecollection get list patch update watch]
services	svc	v1	true	Service	[create delete get list patch update watch]
mutatingwebhookconfigurations		admissionregistration.k8s.io/v1	false	MutatingWebhookConfiguration	[create delete deletecollection get list patch update watch]
validatingwebhookconfigurations		admissionregistration.k8s.io/v1	false	ValidatingWebhookConfiguration	[create delete deletecollection get list patch update watch]
customresourcedefinitions	crd,crds	apirestensions.k8s.io/v1	false	CustomResourceDefinition	[create delete deletecollection get list patch update watch]
apiservices		apiregistration.k8s.io/v1	false	APIService	[create delete deletecollection get list patch update watch]
controllerrevistsions		apps/v1	true	ControllerRevision	[create delete deletecollection get list patch update watch]
daemonssets	ds	apps/v1	true	DaemonSet	[create delete deletecollection get list patch update watch]
deployments	deploy	apps/v1	true	Deployment	[create delete deletecollection get list patch update watch]
replicasets	rs	apps/v1	true	ReplicaSet	[create delete deletecollection get list patch update watch]
statefulsets	sts	apps/v1	true	StatefulSet	[create delete deletecollection get list patch update watch]
tokenreviews		authentication.k8s.io/v1	false	TokenReview	[create]
localsubjectaccessreviews		authorization.k8s.io/v1	true	LocalSubjectAccessReview	[create]

## 7.11 Obtaining API Versions (contd.)

selfsubjectaccessreviews		authorization.k8s.io/v1	false	SelfSubjectAccessReview	[create]
selfsubjectrulesreviews		authorization.k8s.io/v1	false	SelfSubjectRulesReview	[create]
subjectaccessreviews		authorization.k8s.io/v1	false	SubjectAccessReview	[create]
horizontalpodautoscalers	hpa	autoscaling/v1	true	HorizontalPodAutoscaler	[create delete deletecollection get list patch update watch]
cronjobs	cj	batch/v1beta1	true	CronJob	[create delete deletecollection get list patch update watch]
jobs		batch/v1	true	Job	[create delete deletecollection get list patch update watch]
certificatesigningrequests	csr	certificates.k8s.io/v1	false	CertificateSigningRequest	[create delete deletecollection get list patch update watch]
leases		coordination.k8s.io/v1	true	Lease	[create delete deletecollection get list patch update watch]
endpointslices		discovery.k8s.io/v1beta1	true	EndpointSlice	[create delete deletecollection get list patch update watch]
events	ev	events.k8s.io/v1	true	Event	[create delete deletecollection get list patch update watch]
ingresses	ing	extensions/v1beta1	true	Ingress	[create delete deletecollection get list patch update watch]
flowschemas		flowcontrol.apiserver.k8s.io/v1beta1	false	FlowSchema	[create delete deletecollection get list patch update watch]
prioritylevelconfigurations		flowcontrol.apiserver.k8s.io/v1beta1	false	PriorityLevelConfiguration	[create delete deletecollection get list patch update watch]
ingressclasses		networking.k8s.io/v1	false	IngressClass	[create delete deletecollection get list patch update watch]
ingresses	ing	networking.k8s.io/v1	true	Ingress	[create delete deletecollection get list patch update watch]
networkpolicies	netpol	networking.k8s.io/v1	true	NetworkPolicy	[create delete deletecollection get list patch update watch]
runtimeclasses		node.k8s.io/v1	false	RuntimeClass	[create delete deletecollection get list patch update watch]
poddisruptionbudgets	pdb	policy/v1beta1	true	PodDisruptionBudget	[create delete deletecollection get list patch update watch]
podsecuritypolicies	psp	policy/v1beta1	false	PodSecurityPolicy	[create delete deletecollection get list patch update watch]
clusterrolebindings		rbac.authorization.k8s.io/v1	false	ClusterRoleBinding	[create delete deletecollection get list patch update watch]
clusterroles		rbac.authorization.k8s.io/v1	false	ClusterRole	[create delete deletecollection get list patch update watch]
rolebindings		rbac.authorization.k8s.io/v1	true	RoleBinding	[create delete deletecollection get list patch update watch]
roles		rbac.authorization.k8s.io/v1	true	Role	[create delete deletecollection get list patch update watch]
priorityclasses	pc	scheduling.k8s.io/v1	false	PriorityClass	[create delete deletecollection get list patch update watch]
storageclasses		storage.k8s.io/v1	false	StorageClass	[create delete deletecollection get list patch update watch]
storageclasses	sc	storage.k8s.io/v1	false	StorageClass	[create delete deletecollection get list patch update watch]
volumeattachments		storage.k8s.io/v1	false	VolumeAttachment	[create delete deletecollection get list patch update watch]

## 7.12 Stateless Applications

- A stateless application does not preserve its state and saves no data to persistent storage
- All user and session data stays with the client.
- Some examples of stateless applications include web frontends like Nginx, web servers like Apache Tomcat, and other web applications.
- You can create a Kubernetes Deployment to deploy a stateless application on your cluster.
- Pods created by Deployments are not unique and do not preserve their state, which makes scaling and updating stateless applications easier.

## 7.13 Sample Deployment Manifest File

```
# for versions before 1.9 use apps/v1beta2
apiVersion: apps/v1
kind: Deployment
metadata:
  # Unique key of the Deployment instance
  name: deployment-example
spec:
  # 3 Pods should exist at all times.
  replicas: 3
  template:
    metadata:
      labels:
        # Apply this label to pods and default
        # the Deployment label selector to this value
        app: nginx
    spec:
      containers:
        - name: nginx
          # Run this image
          image: nginx:1.10
          ports:
            - containerPort: 80
```

## 7.14 Working with Deployments

- Create a deployment

```
kubectl apply -f <deployment_file>
```

- View deployment manifest

```
kubectl get deployment <deployment_name> -o yaml
```

- Get detailed information about the deployment

```
kubectl describe deployment <deployment_name>
```

- Get all pods

```
kubectl get pods
```

- Get pods that have a certain label assigned

```
kubect1 get pods -l <key>=<value>
```

- **Get pod details**

```
kubect1 describe pod <pod_name>
```

- **Rolling back an update**

```
kubect1 rollout undo deployment <deployment_name>
```

- **Scale a deployment**

```
kubect1 scale deployment <deployment_name> --replicas  
<num_of_replicas>
```

- **Delete a deployment**

```
kubect1 delete deployment <deployment_name>
```

## 7.15 Stateful Applications

- A stateful application requires that its state be saved or persistent.
- Stateful applications use persistent storage, such as persistent volumes, to save data for use by the server or by other users.
- Examples of stateful applications include databases like MongoDB and message queues like Apache ZooKeeper.
- You can create a Kubernetes StatefulSet to deploy a stateful application.
- Pods created by StatefulSets have unique identifiers and can be updated in an ordered, safe way.

## 7.16 Sample Stateful Manifest File

```
apiVersion: apps/v1  
kind: StatefulSet  
metadata:  
  name: [STATEFULSET_NAME]  
spec:  
  serviceName: [SERVICE_NAME]  
  replicas: 3  
  updateStrategy:  
    type: RollingUpdate  
  template:  
    metadata:  
      labels:  
        app=[APP_NAME]  
    spec:  
      containers:
```

```
- name: [CONTAINER_NAME]
  image: ...
  ports:
  - containerPort: 80
    name: [PORT_NAME]
  volumeMounts:
  - name: [PVC_NAME]
    mountPath: ...
volumeClaimTemplates:
- metadata:
  name: [PVC_NAME]
  annotations:
  ...
spec:
  accessModes: [ "ReadWriteOnce" ]
  resources:
    requests:
      storage: 1Gi
```

### 7.17 Sample Stateful Manifest File (Contd.)

- [STATEFULSET\_NAME] is the name you choose for the StatefulSet
- [SERVICE\_NAME] is the name you choose for the Service
- [APP\_NAME] is the name you choose for the application run in the Pods
- [CONTAINER\_NAME] is name you choose for the containers in the Pods
- [PORT\_NAME] is the name you choose for the port opened by the StatefulSet
- [PVC\_NAME] is the name you choose for the PersistentVolumeClaim

### 7.18 Working with StatefulSet

- Create a statefulset

```
kubectl apply -f <statefulset_file>
```
- View statefulset manifest

```
kubectl get statefulset <statefulset_name> -o yaml
```
- Get detailed information about the statefulset

```
kubectl describe statefulset <statefulset_name>
```
- List PersistentVolumeClaims created

```
kubectl get pvc
```

- **Get all pods**  
`kubectl get pods`
- **Get pods that have a certain label assigned**  
`kubectl get pods -l <key>=<value>`
- **Get pod details**  
`kubectl describe pod <pod_name>`
- **Rolling back an update**  
`kubectl rollout undo statefulset <statefulset_name>`
- **Scale a statefulset**  
`kubectl scale statefulset <statefulset_name> --replicas <num_of_replicas>`
- **Delete a statefulset**  
`kubectl delete statefulset <statefulset_name>`

## 7.19 Jobs

- Jobs represent finite, independent, and often parallel tasks which run to their completion.
- Some examples of jobs include automatic or scheduled tasks like sending emails, rendering video, and performing computations.
- You can create a Kubernetes Job to execute and manage a batch task on your cluster.
- You can specify the number of Pods that should complete their tasks before the Job is complete, as well as the maximum number of Pods that should run in parallel.

## 7.20 Sample Job Manifest File

- The following Job computes pi to 2000 places then prints it

```
apiVersion: batch/v1
kind: Job
metadata:
  # Unique key of the Job instance
```



```
name: example-job
spec:
  completions: 1
  activeDeadlineSeconds: 100
  parallelism: 5
  template:
    metadata:
      name: example-job
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl"]
        args: ["-Mbignum=bpi", "-wle", "print bpi(2000)"]
        # Do not restart containers after they exit
        restartPolicy: Never
```

## 7.21 Sample Job Manifest File (Contd.)

- **completions** - it's optional. By default, when a job completes, it terminates the pod. You can specify the number of times the operation should be completed before the pod is terminated.
- **activeDeadlineSeconds**: it's optional. By default, a pod will wait indefinitely for the operation to complete.
- **parallelism**: it's optional. By default, multiple identical jobs cannot run in parallel. You can specify the number of simultaneous jobs that can run at a given time.

## 7.22 Working with Batch Job

- Deploy a job

```
kubectl apply -f <job_file>
```

- View job details

```
kubectl describe job <job_name>
```

- Scale a job

```
kubectl scale job my-job --replicas=<value>
```

- Delete a job

```
kubectl delete job <job_name>
```

## 7.23 DaemonSets

- DaemonSets perform ongoing background tasks in their assigned nodes without the need for user intervention.
- Examples of DaemonSets include log collectors like Fluentd and monitoring services, such as Prometheus
- You can create a Kubernetes DaemonSet to deploy it on your cluster.
- DaemonSets create one Pod per node, and you can choose a specific node to which the DaemonSet should deploy.

## 7.24 DaemonSets (contd.)



source: <https://www.blumatador.com/hs-fs/hubfs/blog/new/An%20Introduction%20to%20Kubernetes%20DaemonSets/DaemonSets.png?width=770&name=DaemonSets.png>

## 7.25 Sample Daemon Manifest File

- The following configuration file is for a daemon that prints the hostname on each Node in the cluster every 10 seconds

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  # Unique key of the DaemonSet instance
  name: daemonset-example
spec:
  template:
    metadata:
      labels:
        app: daemonset-example
    spec:
      containers:
        # This container is run once on each Node in the cluster
        - name: daemonset-example
          image: ubuntu:trusty
          command:
            - /bin/sh
          args:
            - -c
            # This script is run through `sh -c <script>`
            - >-
              while [ true ]; do
                echo "DaemonSet running on $(hostname)" ;
                sleep 10 ;
              done
```

## 7.26 Rolling Updates

- You can perform a rolling update to update the images, configuration, labels, resource limits, and annotations of the workloads in your clusters.
- Rolling updates incrementally replace your resource's Pods with new ones, which are then scheduled on nodes with available resources.
- Rolling updates are designed to update your workloads without downtime.
- The following objects represent Kubernetes workloads.
  - ◇ DaemonSets
  - ◇ Deployments
  - ◇ StatefulSets
- You can trigger a rolling update on these workloads by updating their Pod

Template.

- The Pod Template contains the following fields:
  - ◇ containers: image
  - ◇ metadata: labels
  - ◇ volumes

## 7.27 Rolling Updates (Contd.)

- Change object's image

```
kubectl set image deployment nginx nginx=nginx:1.9.1
```

- Set resource limit

```
kubectl set resources deployment nginx --limits  
cpu=200m,memory=512Mi --requests cpu=100m,memory=256Mi
```

- Check rollout status

```
kubectl rollout status deployment nginx
```

- Pause a rollout

```
kubectl rollout pause deployment nginx
```

- Resume rollout update

```
kubectl rollout resume deployment nginx
```

## 7.28 Rolling Updates (Contd.)

- View rollout history

```
kubectl rollout history deployment nginx
```

- View details of a specific revision

```
kubectl rollout history deployment nginx --revision 3
```

- Rollback an update

```
kubectl rollout undo deployments nginx
```

- Rollback to a specific revision

```
kubectl rollout undo deployment nginx --to-revision 3
```

## 7.29 Summary

- Some common types of workloads and the Kubernetes controller objects you can create include:
  - ◇ Stateless applications
  - ◇ Stateful applications
  - ◇ Batch jobs
  - ◇ Daemons



## Chapter 8 - Scheduling and Node Management

---

### *Objectives*

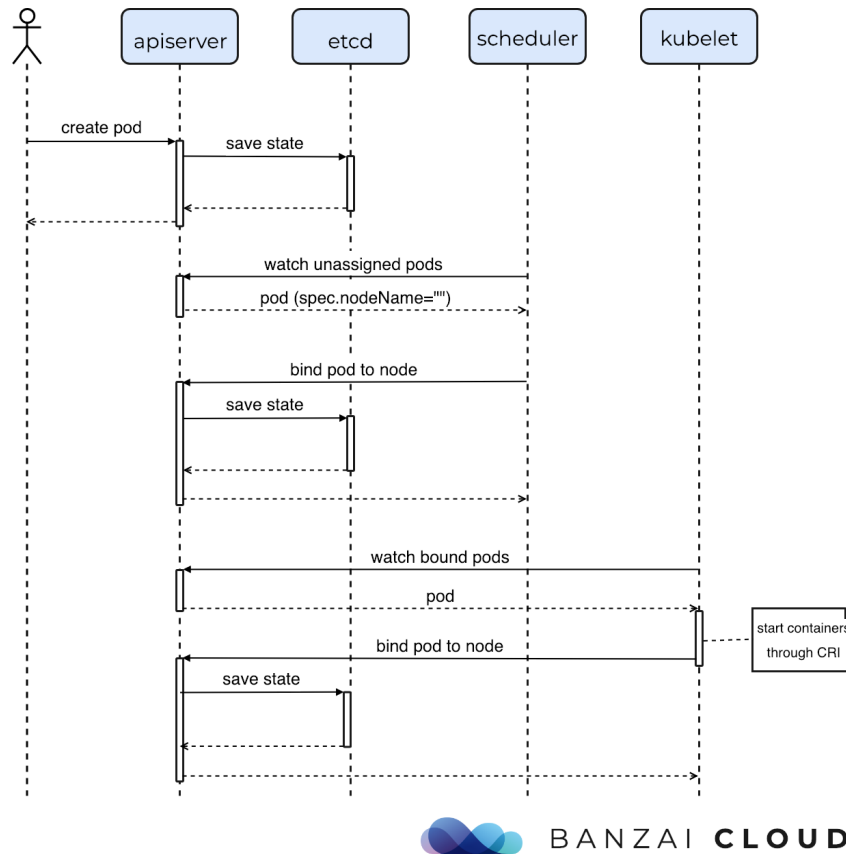
Key objectives of this chapter

- Scheduling Pods to Nodes
- Affinities

### **8.1 Kubernetes Scheduler**

- One of the primary jobs of the Kubernetes is to schedule containers to worker nodes in the cluster of machines.
- The scheduling is accomplished by a component in the Kubernetes cluster called the Kubernetes scheduler.
- Kubernetes can handle a wide variety of workloads, such as web/API application serving, big data batch jobs, and machine learning.
- The key to ensuring that all of these very different applications can operate in on the same cluster lies in the application of job scheduling
- Kubernetes scheduler ensures that each container is placed onto the worker node best suited to it.

## 8.2 Kubernetes Scheduler Overview (contd.)



source: <https://banzaicloud.com/blog/k8s-custom-scheduler/>

## 8.3 Kubernetes Scheduler Overview (contd.)

- In Kubernetes, the nodeName field indicates the node on which a Pod should execute.
- When a Pod is first created, it generally doesn't have a nodeName field.
- The Kubernetes scheduler is constantly scanning for Pods that don't have a nodeName
- The Pods without the nodeName field are eligible for scheduling.
- The scheduler selects an appropriate node for the Pod and updates the Pod definition with the nodeName that the scheduler selected.
- After the nodeName is set, the kubelet running on that node is notified about the Pod's existence and it executes that Pod on that node.



## 8.4 Skip Kubernetes Scheduler

- To skip the scheduler, you can set the nodeName yourself on a Pod.
- This direct schedules a Pod onto a specific node.
- This is how the DaemonSet controller schedules a single Pod onto each node in the cluster.
- In general, direct scheduling should be avoided, since it tends to make your application more brittle and your cluster less efficient.
- You should trust the scheduler to make the right decision, just as you trust the operating system to find a core to execute your program when you launch it on a single machine.

## 8.5 Scheduling Process

- The node for a Pod is determined by several factors, some of which are supplied by the user and the others are calculated by the scheduler.
- The scheduling process relies on the following:
  - ◇ Predicates
  - ◇ Priorities

## 8.6 Scheduling Process - Predicates

- A predicate indicates whether a Pod fits onto a particular node.
- Predicate is a hard constraint, which, if violated, lead to a Pod not operating correctly on that node.
- For example:
  - ◇ the amount of memory requested by the Pod. If that memory is unavailable on the node, the Pod cannot get all of the memory that it needs and the constraint is violated—it is false.
  - ◇ a node-selector label query specified by the user. In this case, the user has requested that a Pod only run on certain machines as indicated by the node labels. The predicate is false if a node does not have the required label.

## 8.7 Scheduling Process - Priorities

- Predicates indicate situations that are either true or false—the Pod either fits or it doesn't
- Priority is an additional generic interface used by the scheduler to determine preference for one node over another.
- The role of priorities or priority functions is to score the relative value of scheduling a Pod onto a particular node.
- Unlike predicates, the priority function does not indicate whether or not the Pod being scheduled onto the node is viable
- The predicate function attempts to judge the relative value of scheduling the Pod onto that particular node.
- For example, a priority function would weight nodes where the image has already been pulled. Therefore, the container would start faster than nodes where the image is not present and would have to be pulled, delaying Pod startup.
- Ultimately, all of the various predicate values are mixed to achieve a final priority score for the node, and this score is used to determine where the Pod is scheduled.

## 8.8 Scheduling Algorithm

- For every Pod that needs scheduling, the scheduling algorithm is run.
- At a high level, the algorithm looks like this:

```
schedule(pod): string
    nodes := getAllHealthyNodes()
    viableNodes := []
    for node in nodes:
        for predicate in predicates:
            if predicate(node, pod):
                viableNodes.append(node)

    scoredNodes := PriorityQueue<score, Node[]>
    priorities := GetPriorityFunctions()
    for node in viableNodes:
        score = CalculateCombinedPriority(node, pod, priorities)
        scoredNodes[score].push(node)

    bestScore := scoredNodes.top().score
```

```
selectedNodes := []
while scoredNodes.top().score == bestScore:
    selectedNodes.append(scoredNodes.pop())

node := selectAtRandom(selectedNodes)
return node.Name
```

- The actual code is available on the Kubernetes GitHub page.

<https://github.com/kubernetes/kubernetes>

### 8.9 Kubernetes Scheduling Algorithm

- The scheduler gets the list of all currently known and healthy nodes.
- For each predicate, the scheduler evaluates the predicate against the node and the Pod being scheduled.
- If the node is viable, the node is added to the list of possible nodes for scheduling.
- All of the priority functions are run against the combination of Pod and node.
- The results are pushed into a priority queue ordered by score, with the best-scoring nodes at the top of the queue.
- All nodes that have the same score are popped off of the priority queue and placed into a final list.
- One of the nodes is chosen in a round-robin fashion and is then returned as the node where the Pod should be scheduled.
- Round robin is used instead of random choice to ensure an even distribution of Pods among identical nodes.

### 8.10 Scheduling Conflicts

- There is lag time between when a Pod is scheduled (time T1) and when the container executes (time T2)
- Due to the lag time, the scheduling decision may become invalid.
- In some cases, this may mean that a slightly less ideal node is chosen, when a better one could have been assigned.
- In general, these sorts of conflicts aren't that important and they normalize

in the aggregate.

- These conflicts are thus ignored by Kubernetes.
- In case of more severe conflicts, when the node notices that it has been asked to run a Pod that no longer passes the predicates for the Pod and node, the Pod is marked as failed.
- The failed Pod doesn't count as an active member of the ReplicaSet and, thus, a new Pod will be created and scheduled onto a different node where it fits.

**Note:**

There is some work going on in the Kubernetes community to improve this situation somewhat. A Kubernetes-descheduler project, which, if run in a Kubernetes cluster, scans it for Pods that are determined to be significantly suboptimal. If such Pods are found, the descheduler evicts the Pod from its current node. Consequently, the Pod is rescheduled by the Kubernetes scheduler, as if it had just been created.

## 8.11 Controlling Scheduling

- Adding custom predicates and priorities is a fairly heavyweight task.
- Kubernetes provides you with several tools to customize scheduling—without having to implement anything in your code
  - ◇ Labels
  - ◇ Affinity
  - ◇ Taints
  - ◇ Tolerations

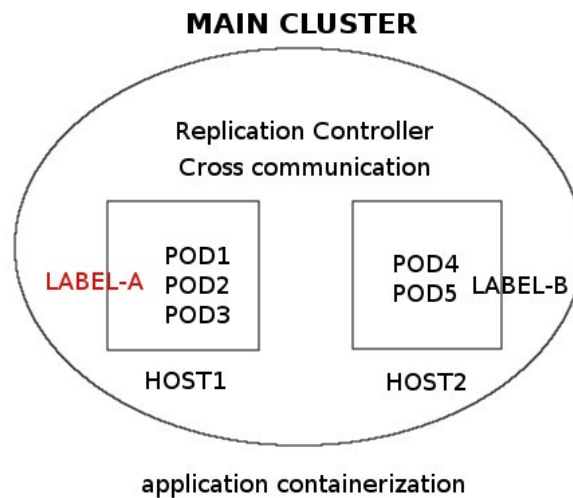
## 8.12 Label Selectors

- Every object in Kubernetes has an associated set of labels.
- Labels provide identifying metadata for Kubernetes objects, and label selectors are often used to dynamically identify sets of objects for various operations.
- Label selectors can also be used to identify a subset of the nodes in a

Kubernetes cluster that should be used for scheduling a particular Pod.

- By default, all nodes in the cluster are potential candidates for scheduling, but by filling in the `spec.nodeSelector` field in a Pod or PodTemplate, the initial set of nodes can be reduced to a subset.

### 8.13 Label Selectors (contd.)



source: <http://network-insight.net/2016/03/kubernetes-container-scheduler/>

### 8.14 Label Selectors (Contd.)

- Add a label to a node:

```
kubectl label nodes <node-name> <label-key>=<label-value>
```

- Add a `nodeSelector` field to your pod configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod-name>
spec:
  containers:
  - name: <container-name>
    image: <image-name>
nodeSelector:
```

**<label-key>: <label-value>**

- Schedule the pod on the node

```
kubectl apply -f <pod-config-yaml>
```

## 8.15 Node Affinity and Anti-affinity

- Node selectors provide a simple way to guarantee that a Pod lands on a particular node, but they lack flexibility.
- They cannot represent more complex logical expressions (e.g., “Label 1 is either A or B.”) nor can they represent anti-affinity (“Label 1 is A but label 2 is not C.”).
- For example, in a three-node cluster, a web application has in-memory cache such as redis. We want the web-servers to be co-located with the cache.
- Affinity is a more complicated structure to understand, but it is significantly more flexible if you want to express more complicated scheduling policies.
- You can use operators, such as In, NotIn, Exists, NotExists, Gt, and Lt

## 8.16 Node Affinity Example

- Consider the example just noted, in which a Pod should schedule onto a node that has label 1 with a value of either A or B. This is expressed as the following affinity policy:

```
kind: Pod
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              # label_1 == A or B
            - key: label_1
              operator: In
```

```
        values:
        - A
        - B
    ...
```

## 8.17 Node Antiaffinity Example

- To show antiaffinity, consider the policy label 1 has value A and label 2 does not equal C. This is expressed in a similar specification:

```
kind: Pod
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          # label_1 == A
          - key: label_1
            operator: In
            values:
            - A
          # label_2 != C
          - key: label_2
            operator: NotIn
            values:
            - C
    ...
```

## 8.18 Taints and Tolerations

- Node affinity is a property of pods that attracts them to a set of nodes
- Taints are the opposite as they allow a node to repel a set of pods.
- Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes.
- One or more taints are applied to a node that means the node should not accept any pods that do not tolerate the taints.

- Tolerations are applied to pods, and allow the pods to schedule onto nodes with matching taints.

## 8.19 Taints and Tolerations (Contd.)

- You add a taint to a node using `kubectl taint`.

```
kubectl taint nodes node1 key=value:<Effect>
```

- You can use the following effects:
  - ◇ **NoSchedule** - no pod will be able to schedule on to the node
  - ◇ **PreferNoSchedule** - a “preference” or “soft” version of NoSchedule – the system will try to avoid placing a pod that does not tolerate the taint on the node, but it is not required.
  - ◇ **NoExecute** - the pod will be evicted from the node (if it is already running on the node), and will not be scheduled onto the node (if it is not yet running on the node).

- To remove the taint, run the following command:

```
kubectl taint nodes node1 key1=value1:NoSchedule-
```

- To remove all taints from a node, run the following command:

```
kubectl patch node node1.compute.internal -p '{"spec": {"taints":[]}}'
```

## 8.20 Taints and Tolerations (Contd.)

- Sample Toleration in a pod configuration:

```
tolerations:  
- key: "key"  
  operator: "Equal"  
  value: "value"  
  effect: "NoSchedule"
```

## 8.21 Taints and Tolerations - Example

- If the node is tainted like this:



```
kubectl taint nodes node1 key1=value1:NoSchedule
kubectl taint nodes node1 key1=value1:NoExecute
kubectl taint nodes node1 key2=value2:NoSchedule
```

- ... and a pod has the following tolerations:

```
tolerations:
```

```
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

- ... the pod will not be able to schedule onto the node, because there is no toleration matching the third taint. But it will be able to continue running if it is already running on the node when the taint is added because the third taint is the only one of the three that is not tolerated by the pod.

## 8.22 Summary

- Kubernetes scheduler is a component of Kubernetes that decides on what node a pod should execute.
- To gain fine-control over the scheduling, labels, affinity, taints, and tolerations can be used.



## Chapter 9 - Managing Networking

---

### *Objectives*

Key objectives of this chapter

- Networking Components
- Kubernetes Network Model
- Networking Scenarios
- Container to Container Communication
- Pod to Pod Communication
- Pod to Service Communication
- External to Service Communication
- Accessing Applications
- CNI

### **9.1 Kubernetes Networking Components**

- Containers - Application execution environments
- Pods - Pods hold containers
- Service - Collection of Pods grouped for networking purposes
- Worker Nodes - Pods are executed on worker nodes
- Master Node - Manages worker nodes
- External Network - the network running outside the K8s cluster
- External Applications - applications that access K8s clustered applications/services or that are accessed by K8s clustered applications/services

### **9.2 The Kubernetes Network Model**

- Pods are treated like physical or VM hosts
- Each Pod gets its own IP address
- Containers within Pods are expose ports

- Containers in a Pod can reach each other's ports on localhost much like multiple applications running on a common physical machine or VM
- Services expose Pods to the external network
- Kubernetes keeps the list of endpoints current as Pods and Services are created and destroyed
- The Kubernetes API allows for discovery of the endpoints it keeps

## Notes

For more information see: <https://kubernetes.io/docs/concepts/cluster-administration/networking/#the-kubernetes-network-model>

## 9.3 Networking Scenarios

- Networking in Kubernetes boils down to these scenarios:
  - ◇ Container-Container communication
  - ◇ Pod-Pod communication
  - ◇ Pod-Service communication
  - ◇ External-to-Service communication

## Notes

## 9.4 Container-Container Communication

- Pod acts as the Host
- Containers expose ports on the host
- Container port allocation must be aware of other containers
- Apps in the Pod's containers contact each other over localhost
- Containers in the Pod share resources(vol, cpu, ram, etc.)
- Containers are therefore not 100% isolated from each other, which should be OK in some cases

- When total isolation is needed an alternative setup may be used, for example placing containers in Pods on different nodes

## Notes

### 9.5 Pod-Pod Communication

- Pods communicate with one another based on their IP addresses
- Each Pod has a real IP address
- Pod IP addresses are not simply addresses private to the node

## Notes

### 9.6 1.3 Pod-Service Communication

- Services allow grouping of Pods for load balancing purposes
- Services create a virtual IP address that acts as a single point of connection to its member Pods
- Individual connections are redirected as they come in to one of the service's Pods based on load balancing parameters

## Notes

### 9.7 External-Service Communication

- Use a Kubernetes Service to connect to nodes, pods and other services from outside the k8s cluster
- Service type should be NodePort or LoadBalancer
- Creating services can be done in two ways:

- ◇ Use the 'kubectl expose ...' command
- ◇ Use the 'kubectl apply ...' command and a service spec (\*.yaml)
- Expose can create service based on pods, services, replicationcontrollers, deployments, replicaset
- Services can address a specific pod based on an applied label for debugging and troubleshooting purposes

## Notes

### 9.8 Accessing Applications

- Applications can be accessed in a variety of ways:
  - ◇ Shell into the app's container and use localhost
  - ◇ Shell into any container in the same pod as the app and again use localhost to access the app
  - ◇ Apps on the same node can access apps in different pods using the app pod's name or IP address
  - ◇ Apps outside a cluster can access a service in a k8s cluster through a service-defined port on the cluster's IP address.
  - ◇ The 'kubectl port-forward' command can be used to forward a port on the local machine where the cluster is running to a pod running inside the cluster.

## 9.9 Useful Commands

Command	Usage
<code>minikube ip</code>	Output cluster IP address
<code>minikube service {service-name}</code>	Open service-name service in host browser
<code>minikube service --url=true {service-name}</code>	Output external ip address for service-name
<code>kubectl cluster-ip</code>	Outputs URLs for Master node and KubeDNS
<code>curl {url}</code>	OS command for making HTTP calls
<code>ping {ip-address}</code>	OS command for testing server existence
<code>ifconfig (linux)</code>	Linux/mac cmd displaying network config
<code>ipconfig (win)</code>	Windows cmd displaying network config

## 9.10 Container Network Interface (CNI)

- Container Network Interface is a Kubernetes plugin
- The CNI plugin is responsible for inserting a network interface into the container network namespace
  - ◇ For example, one end of a virtual ethernet pair
- It also makes any necessary changes on the host
  - ◇ For example, attaching the other end of the veth into a bridge
- The CNI plugin assigns an IP address to the interface and configures the routes by invoking the appropriate IP Address Management (IPAM) plugin.
- CNI is used when you use a non-Docker, or OCI (Open Container Initiative), runtime such as CRI-O and Containerd.

## 9.11 What is CNI's Role?

- CNI is used by container runtimes, such as Kubernetes, Podman, CRI-O, Mesos, and others.
- When a container/pod is created, it has no network interface.
- The container runtime calls the CNI plugin with verbs such as ADD, DEL, CHECK, etc.
- The ADD verb creates passes the network interface details via JSON payload to create a new network interface for the container

## 9.12 CNI Configuration Format

- **cniVersion (string)** - currently "1.0.0"
- **name (string)** - network name
- **disableCheck (boolean)** - whether runtimes should perform a check for the network configuration list.
- **plugins (list)** - a list of CNI plugins and their configuration
- type (string) - CNI plugin binary
- capabilities
- **dns (dictionary)** - a dictionary that can contain the following:
  - ◇ **nameservers (list of strings)**- a priority-ordered list DNS nameservers
  - ◇ **domain (string)** - the local domain used for short hostname lookups
  - ◇ **options (list of strings)** - list of options that can be passed to the resolver.
- Details are available on the official CNI project page:  
<https://github.com/containernetworking/cni/blob/master/SPEC.md#Deriving-runtimeConfig>

## 9.13 Sample CNI Configuration

```
{  
  "cniVersion": "1.0.0",  
  "name": "dbnet",  
}
```



```
"disableCheck": "false",
"plugins": [
  {
    "type": "bridge",
    // plugin specific parameters
    "bridge": "cni0",

    "ipam": {
      "type": "host-local",
      // ipam specific
      "subnet": "10.1.0.0/16",
      "gateway": "10.1.0.1",
      "routes": [
        {"dst": "0.0.0.0/0"}
      ]
    },
    "dns": {
      "nameservers": [ "10.1.0.1" ]
    }
  },
  {
    "type": "tuning",
    "capabilities": {
      "mac": true,
    },
    "sysctl": {
      "net.core.somaxconn": "500"
    }
  },
  {
    "type": "portmap",
    "capabilities": {"portMappings": true}
  }
]
```

### 9.14 Running the CNI Plugins

- Download the plugin binaries from <https://github.com/containernetworking/plugins/releases>
- Download the plugin execution scripts from <https://github.com/containernetworking/cni>
- Create a sample CNI configuration file. (/etc/cni/net.d/ is the default location in which the scripts will look for net configurations)

```
cat >/etc/cni/net.d/99-loopback.conf <<EOF
{
    "cniVersion": "0.2.0",
    "name": "lo",
    "type": "loopback"
```

```
}  
EOF
```

- Execute `priv-net-run.sh` located in the scripts folder of the cloned CNI repository.

```
priv-net-run.sh ifconfig (runs the ifconfig command in a private network namespace  
that has joined the mynetwork network)
```

- To run a docker container with network namespace set up by CNI plugins, run the following script from the CNI repository's script folder.

```
docker-run.sh --busybox:latest ifconfig
```

### 9.15 Summary

In this chapter we covered:

- Networking Components
- Kubernetes Network Model
- Networking Scenarios
- Container to Container Communication
- Pod to Pod Communication
- Pod to Service Communication
- External to Service Communication
- Accessing Applications
- Useful Commands

## Chapter 10 - Managing Persistent Storage

---

### *Objectives*

Key objectives of this chapter

- Storage Methods
- Volume Types
- Cloud Resource Types
- configMaps
- emptyDir
- Persistent Volumes and Claims
- Secrets
- Security Contexts

### 10.1 Storage Methods

- Storage Methods
  - ◇ Container OS file system storage
  - ◇ Docker Volumes
  - ◇ Kubernetes Volumes

### Notes

### 10.2 Container OS file system storage

- Containers run operating system images that include file systems.
- It is possible to create or modify files in the OS image fs and have them used by applications.
- This can cause problems though If container crashes. Any files added/modified since it was started will be lost

- Another downside to using the image fs for file storage is that files in the container can't be shared with other containers if needed.

## Notes

### 10.3 Docker Volumes

- The Docker container manager includes storage volumes.
- Docker storage volumes:
  - ◇ Consist of a directory on disk with some data
  - ◇ Do not have a managed lifecycle
  - ◇ Can use volume drivers but they are of limited functionality
  - ◇ Impose a limit of one vol driver per container

## Notes

### 10.4 Kubernetes Volumes

- Kubernetes has its own implementation of volumes that:
  - ◇ Also consist of a directory on disk with some data
  - ◇ Has its volumes created in Pods
  - ◇ Have a lifetime tied to Pod that encloses them
  - ◇ Allows data to be persisted across container restarts
  - ◇ Supports many volume types through 'volume drivers'
  - ◇ Lets Pods use multiple volumes at the same time

## Notes

## 10.5 K8S Volume Types

- A few examples of Kubernetes volume are listed here:
  - ◇ `awsElasticBlockStore`
  - ◇ `azureDisk`
  - ◇ `azureFile`
  - ◇ `gcePersistentDisk`
  - ◇ `configMap`
  - ◇ `emptyDir`
  - ◇ `hostPath`
  - ◇ `local`
  - ◇ `nfs`

### Notes

## 10.6 Cloud Resource Types

- Cloud Resource Types
  - ◇ Must be created on cloud platform before being used in K8s
  - ◇ Volume is independent of Pod, when Pod goes away the volume is simply unmounted and not deleted
  - ◇ Pods typically need to be running on the cloud provider
- `awsElasticBlockStore`
  - ◇ Mounts Amazon Web Services(AWS) EBS volumes into a Pod.
  - ◇ Can only be used with Pods that are running on AWS EC2 nodes
- `azureDisk` & `azureFile`

- ◇ An azureDisk is used to mount a Microsoft Azure Data Disk into a Pod.
- ◇ An azureFile is used to mount a Microsoft Azure File Volume (SMB 2.1 and 3.0) into a Pod.
- gcePersistentDisk
  - ◇ Mounts a Google Compute Engine (GCE) Persistent Disk into Pod.

## 10.7 configMaps

- Provides a way to inject configuration data into Pods
- Data consists of names and values:

```
color.text=black
color.background=lightgrey
```

- configMaps must be created before they can be used.
- configMaps can be created from literals or files

```
kubectl create configmap my-config --from-file=path
```

- configMaps are mounted and accessed as volumes

```
volumes:
- name: foo
  configMap:
    name: myconfigmap
```

## Notes

For more info on configMaps see the following link:

<https://kubernetes.io/docs/concepts/configuration/configmap/>

## 10.8 Creating configMaps from Literals

- configMaps can be created from literals
- This is a name/value pair we want to access

```
background=lightgrey
```

- Execute this command:

```
kubectl create configmap config1 --from-literal=background=lightgrey
Multiple 'from-literal' sections can be added in the same command
```

- Verify that the configMap was created:

```
kubectl describe configmap config1
```

### 10.9 Creating configMaps from files

- Create a directory for the config file

```
mkdir configs
```

- Create a file named 'myconfig' in the above dir with these values:

```
color.text=black
color.background=lightgrey
```

- Execute this command to create the configMap

```
kubectl create configmap myconfig --from-file=configs
```

- Verify that the configMap was created

```
kubectl describe configmap myconfig
```

### 10.10 Using configMaps

- This Pod definition creates a pod named myconfig that mounts the configMap named "myconfig"

```
# myconfig-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: myconfig
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: config
      mountPath: "/etc/config"
      readOnly: true
  volumes:
```

```
- name: config
  configMap:
    name: myconfig
```

- Create the pod and open a shell into it and access the config file:

```
kubectl apply -f myconfig-pod.yaml
kubectl exec -it myconfig -- bash
cat /etc/config/myconfig
```

## 10.11 emptyDir

- Used to hold temporary files
- An emptyDir volume is created when a Pod is assigned to a Node
- Is deleted when a Pod is removed
- The dir exists as long as the Pod is running on a specific node
- Any container in the Pod can read and write to the emptyDir volume

### Notes

## 10.12 Using an emptyDir Volume

- Pod Specification

```
# empty-dir-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: empty-dir-pod
spec:
  containers:
    - image: nginx:latest
      name: empty-dir-container
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

- Notice how the 'cache-volume' is created and then used in the



## volumeMount

- The empty volume is available at the mountPath '/cache' inside the container

## Notes

Full instructions for creating the pod and investigating the volume

```
# empty-dir-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: empty-dir-pod
spec:
  containers:
    - image: nginx:latest
      name: empty-dir-container
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}

kubectl apply -f empty-dir-pod.yaml
kubectl describe pods empty-dir-pod
kubectl get pods
kubectl exec -it empty-dir-pod -- bash
/# ls cache
/# echo "some file content" > cache/file1.txt
/# cat cache/file1.txt
some file content
exit
```

## 10.13 Other Volume Types

- **hostPath** - A hostPath volume mounts a file or directory from the host node's filesystem into your Pod. Any given hostPath may no longer be available if a Pod get re-scheduled to a different node.
- **local Volume** - A local volume represents a mounted local storage device such as a disk, partition or directory. Is know to the scheduler so pods using it always gets placed on the correct node.
- **nfs Volume** - An nfs volume allows an existing NFS (Network File System) share to be mounted into your Pod.

## Notes

### 10.14 Persistent Volumes

- persistentVolumeClaims allow pods to request storage space without knowing how or from where the storage will be provided
- The following K8s objects are created separately when setting up a Persistent Volume
  - ◇ PersistentVolume (pv) - Defines a volume
  - ◇ PersistentVolumeClaim (pvc) - Defines a need for persistent vol space
  - ◇ Pod - Uses a claim to request a portion of volume space from k8s
- PVCs are associated with PVs through storageClassName, they simply request a type and size of storage needed
- K8s picks the PV to use by storageClassName and resource availability
- The same PV can supply space to multiple Pods
- K8s decides which node to place the PV on based on its nodeAffinity settings

### 10.15 Creating a Volume

- The following operations are required when setting up a Pod to use a Persistent Volume
  - ◇ Create a yaml file for
    - The claim (lpvc.yaml)
    - The persistent volume (lpv.yaml)
    - The Pod (lvppod.yaml)
  - ◇ Apply each yaml file in the above order
  - ◇ K8s will detect that your claim(pvc) and volume (pv) are related and bind them.
  - ◇ K8s will provide storage space from the volume (pv) for the pod to use.

## Notes

The yaml names above can be anything you want. In practice it is helpful to name them in such a way that you can easily determine their purpose from the name later on.

We will look at examples of the three yaml files in the next few slides

## 10.16 Persistent Volume Claim

- Specification file: lpvc.yaml

```
# lpvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: lpv-claim
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

- Commands to apply and check:

```
kubectl apply -f lpvc.yaml
kubectl get pvc
```

- The above spec requests 1Gi of local persistent storage from K8s

## Notes

More information about Persistent Volume Claims can be found here:

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

## 10.17 Persistent Volume

- Specification file: lpv.yaml

```
#lpv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-volume
```

```
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  ...
```

- **Commands to apply and check:**

```
kubectl apply -f lpv.yaml
kubectl get pv
```

- **The above defines 2Gi of local persistent storage on a node's filesystem**

## Notes

More information about PersistentVolumes can be found here:

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

// Full Yaml file, including nodeAffinity

```
#lpv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-volume
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

## 10.18 Pod that uses Persistent Volume

- Specification file: lvpod.yaml

```
# lvpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: lpv-storage
      persistentVolumeClaim:
        claimName: lpv-claim
  containers:
    - name: lpv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: lpv-storage
```

- Commands to apply and check:

```
kubectl apply -f lvpod.yaml
kubectl get pods
```

- The above spec requests storage using the lpv-claim and makes it available at a mount point within the image

## 10.19 Secrets

- Secrets are used to store and manage sensitive information like passwords, tokens and keys.
- Secrets provide an alternative to placing sensitive information in a Pod definition or a container image.
- Pods can consume secrets:
  - ◇ From files in a mounted volume
  - ◇ A container environment variables
- Pods can be created from private images that are pulled by the kubelet. In this case the required secret comes from the imagePullSecrets field

## Notes

### 10.20 Creating Secrets from Files

- Create a secret from a directory of files:
  - ◇ Create a directory for the secrets
  - ◇ In the secrets dir create file whose name is the secret's name and whose contents is the value of the secret

```
Filename: userid  
Contents: jane@abc.com
```

- ◇ Create another secrets file:

```
Filename: password  
Contents: p@ssW0rd
```

- ◇ Run this command:

```
kubectl create secret generic db-creds-1 --from-file=secrets
```

- ◇ Check the secrets

```
kubectl get secrets db-creds-1 -o=json
```

### 10.21 Creating Secrets from Literals

- Create a secret by passing the data to the create command:
  - ◇ These are the secrets we want to save

```
userid=jane@abc.com  
password=$secretp@ss
```

- ◇ Run this command:

```
kubectl create secret generic db-creds-2 --from-  
literal=userid=jane@abc.com --from-literal=password=$secretp@ss
```

- ◇ Check the secrets

```
kubectl get secrets db-creds-2 -o=json
```

## 10.22 Using Secrets

- Pod spec that mounts the secret as a volume

```
# nginx-dbcreds-pod.yaml (partial)
volumeMounts:
  - name: dbcredsvol
    mountPath: "/tmp/dbcreds"
    readOnly: true

volumes:
  - name: dbcredsvol
    secret:
      secretName: db-creds-1
```

- Create the pod and open a shell into it

```
kubectl apply -f nginx-dbcreds-pod.yaml
kubectl exec -it consumesec -c shell -- bash
```

- Access the secret from its mount point

```
cat /tmp/dbcreds/userid
jane@abc.com
```

### Notes

```
// Pod definition - Mount secret as volume
apiVersion: v1
kind: Pod
metadata:
  name: consumesec
spec:
  containers:
    - name: shell
      image: nginx:latest
      command:
        - "bin/bash"
        - "-c"
        - "sleep 10000"
      volumeMounts:
        - name: dbcredsvol
          mountPath: "/tmp/dbcreds"
          readOnly: true
  volumes:
    - name: dbcredsvol
      secret:
        secretName: db-creds-1
```

## 10.23 Security Context

- A security context defines privilege and access control settings for Pods and Containers.

- securityContext settings at the pod level apply to all containers in the pod as well.
- There are many aspects of security that the context can set. Some examples include:
  - ◇ runAsUser
  - ◇ runAsGroup
  - ◇ fsGroup
  - ◇ Linux Capabilities
  - ◇ readOnlyRootFilesystem
  - ◇ ...

## 10.24 Security Context Usage

- Usage:
  - ◇ Design a security context

```
securityContext:
  runAsUser: 1000
  runAsGroup: 3000
  fsGroup: 2000
```
  - ◇ Add the context to a pod specification

```
#security-context-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
```
  - ◇ Create the pod



```
kubectl apply -f security-context.yaml
```

- ◇ The created pod will have the context applied

### Notes

## 10.25 Summary

In this chapter we covered:

- Storage Methods
- Volume Types
- Cloud Resource Types
- configMaps
- emptyDir
- Persistent Volumes and Claims
- Secrets
- Security Contexts



## Chapter 11 - Working with Helm

---

### *Objectives*

Key objectives of this chapter

- What is Helm?
- Installing Helm
- Helm Features
- Helm Terminology
- Searching for Charts
- Adding Repositories
- Installing Charts
- Upgrading and Rolling back Releases
- Creating Custom Charts
- Helm Templates
- Packaging and Installing Custom Charts

### **11.1 What is Helm?**

- Helm is a package manager for Kubernetes
  - ◇ Kubernetes lets you manage application containers
  - ◇ Helm lets you find, install, run and maintain multiple Kubernetes applications as a single unit
- Helm lets anyone run a complex packages even if they are not experts in all the included technologies.
- Helm is used by running commands against the Helm Common Line Interface (CLI)
- Helm Hub at <https://hub.helm.sh> is a web site/internet service for storing and retrieving Helm charts(packages)

## Notes

### 11.2 Installing Helm

- Before installing Helm, Kubernetes and kubectl should be installed
- There are multiple ways to install Helm.
  - ◇ By archive file - download and unzip the archive
  - ◇ By install script - download and run the script
  - ◇ By package manager - Homebrew(mac), Chocolatey(win), apt, yum, Snap, etc.
- Check if Helm is installed by checking its version:
- Development builds (not for production) are also available at [get.helm.sh](https://get.helm.sh)
- For more information on installing Helm see:

```
helm version
```

```
https://helm.sh/docs/intro/install/
```

## Notes

Example Script install:

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

Example Apt-get install:

```
curl https://baltocdn.com/helm/signing.asc | sudo apt-key add - sudo apt-get
install apt-transport-https --yes
```

```
echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee
/etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

## 11.3 Helm and KUBECONFIG

- The Helm client learns about Kubernetes clusters by using files in the Kube config file format
- Helm attempts to find this file at (\$HOME/.kube/config)
- Example kube config contents:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /home/wasadmin/.minikube/ca.crt
    server: https://192.168.99.100:8443
    name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
    name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /home/wasadmin/.minikube/client.crt
    client-key: /home/wasadmin/.minikube/client.key
```

## 11.4 Helm Features

- Helm features include:
  - ◇ Bundling multiple Kubernetes apps
  - ◇ Publishing bundles of apps
  - ◇ Sharing app bundles
  - ◇ Updating app bundles
  - ◇ Rollback to earlier versions of an app bundle

## Notes

### 11.5 Helm Terminology

- **Helm** - The name of the project is capitalized
- **helm** - The Helm CLI client is all lower case
- **Chart** - Kubernetes package, like a multi-app installation blueprint
- **Repository** - A place where you can get and save Charts
- **Release** - A specific instance of a chart running on Kubernetes

## Notes

### 11.6 Searching for Charts with helm CLI

- Install a chart repository:  

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```
- Use the helm search command:
  - ◇ Search a repository:  

```
helm search repo {search-term}
```
  - ◇ Search the Helm Hub  

```
helm search hub {search-term}
```
- Example:

**helm search repo nginx**

NAME	CHART VERSION	APP VERSION	DESCRIPTION
pnnl-miscscripts/nginx-app	0.1.1	0.1.1	A Simple Web service Chart

- Use the 'NAME' to install the chart

```
helm install my-nginx pnnl-miscscripts/nginx-app
```

## Notes

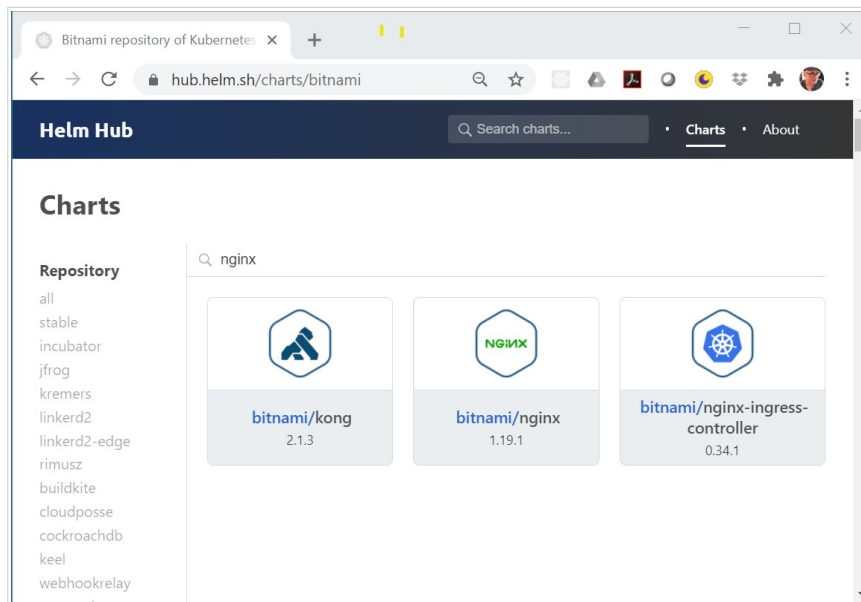
If minikube is not running when you try to install a chart you will get a "Kubernetes cluster unreachable" error.

## 11.7 Adding Repositories

- The "helm search repo ..." command only searches repositories that have been added to the helm locally.
- When helm is first installed it does not contain any repositories
- Some helm repository commands:
  - ◇ `helm repo add {repo-url}` - adds a repository to the local list
  - ◇ `helm repo list` - shows repositories that have been added
  - ◇ `helm repo update` - updates local repositories
  - ◇ `helm repo remove {repo-name}` - removes the named repository

## 11.8 Helm Hub - Search

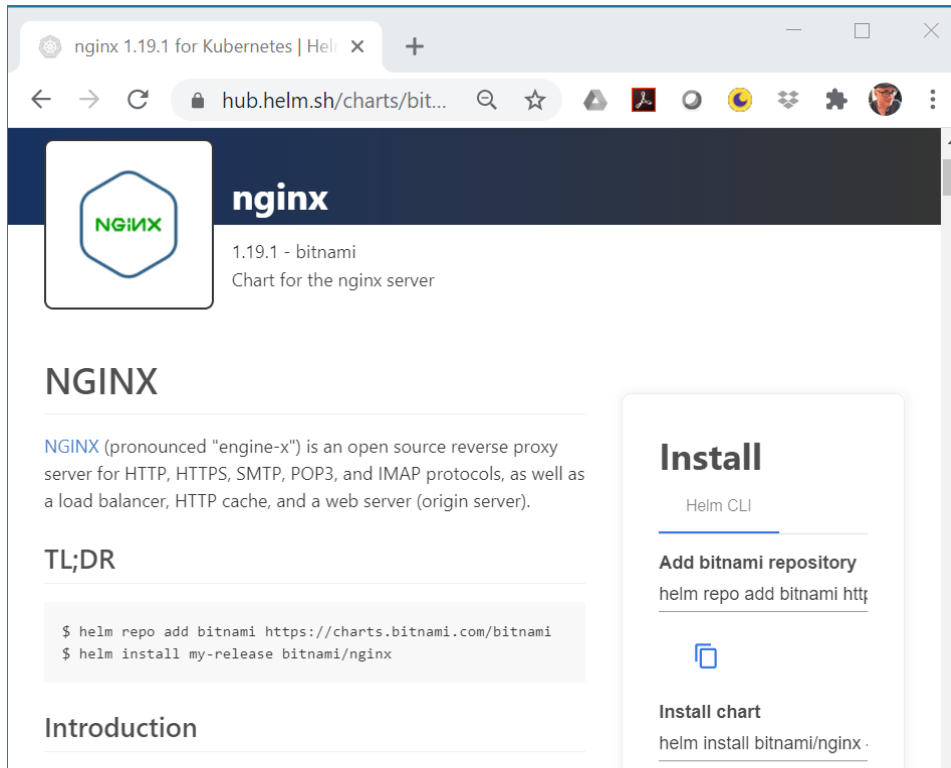
- Charts can also be found at Helm Hub <https://hub.helm.sh/>



## 11.9 Helm Hub - Chart Page

- After searching and choosing a chart a page explaining the chart along with instructions for installing the chart are displayed

```
helm install bitnami/nginx --version 6.1.0
```



## 11.10 Installing a Chart

- helm CLI chart installation command:

```
helm install {release-name} {chart-repo}/{chart-name}
```

- Example (add repo before running install):

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

```
helm install mynginx bitnami/nginx --version 6.1.0
```

- Get status of release



```
helm status mynginx
NAME: mynginx
LAST DEPLOYED: Tue Sep  1 06:37:27 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
...
```

- Remove the release

```
helm uninstall mynginx
```

### 11.11 Upgrading a Release

- Install a release and name it 'mynginx'

```
helm install mynginx bitnami/nginx --version 5.0.0
```

- List the release

```
helm list
NAME      NAMESPACE  REVISION  UPDATED           STATUS  CHART          APP VERSION
mynginx   default     1         2020-09-01 07:50... deployed  nginx-5.0.0    1.16.1
```

- Upgrade the mynginx release

```
helm upgrade mynginx bitnami/nginx --version 6.0.0
```

```
helm list
NAME      NAMESPACE  REVISION  UPDATED           STATUS  CHART          APP VERSION
mynginx   default     2         2020-09-01 07:55... deployed  nginx-6.0.0    1.19.0
```

- The revision number increased to 2 and we also see that the 'nginx-6.0.0' chart included the 1.19.0 version of nginx

## Notes

### 11.12 Rolling Back a Release

- Install and upgrade the 'mynginx' release

```
helm install mynginx bitnami/nginx --version 5.0.0
helm upgrade mynginx bitnami/nginx --version 6.0.0
```

```
helm list
NAME      NAMESPACE  REVISION  UPDATED           STATUS  CHART          APP VERSION
mynginx   default     2         2020-09-01 07:55... deployed  nginx-6.0.0    1.19.0
```

- Rollback the release

```
helm rollback mynginx 1
```

```
helm list
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
mynginx	default	3	2020-09-01 07:50...	deployed	nginx-5.0.0	1.16.1

- The revision number always increases. After rolling back we are at revision 3 of the mynginx release. Revision 3 has the same state as revision 1 did.

## 11.13 Creating Custom Charts

- Start creating a custom chart by executing this command:

```
helm create {chart-name}
```

- Example:

```
helm create my-app
```

- This creates the myapp directory inside the current directory

- The contents of the myapp dir:

```
./my-app
```

```
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   ├── tests
│   └── test-connection.yaml
└── values.yaml
```

### Notes

chart names should be lower case, words in names are separated by a dash.

## 11.14 Common Chart Files

- Some common Chart files and usage:

Filename	Usage
Chart.yaml	Holds version numbers
Values.yaml	Holds variables that are used in templates
deployment.yaml	Template for creating a K8s deployment yaml
hpa.yaml	Template for creating a K8s horizontal pod autoscaler yaml
ingress.yaml	Template for creating a K8s ingress yaml
service.yaml	Template for creating a K8s service yaml
serviceaccount.yaml	Template for creating a K8s service account yaml

## Notes

### 11.15 Helm Templates

- Helm templates are yaml files that Helm uses, along with variables from values.yaml, to create Kubernetes yaml files that can be used to create objects like deployments and services in Kubernetes.
- Helm templates include tags for items that should be replaced when Helm processes the template
- Example from chart deployment.yaml template:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "my-nginx-app.fullname" . }}
  labels:
    {{- include "my-nginx-app.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
```

- The above template fragment results in the following content in the K8s deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: RELEASE-NAME-my-nginx-app
  labels:
```

```
helm.sh/chart: my-nginx-app-0.1.0
app.kubernetes.io/name: my-nginx-app
app.kubernetes.io/instance: RELEASE-NAME
app.kubernetes.io/version: "1.16.0"
app.kubernetes.io/managed-by: Helm
spec:
  replicas: 1
  selector:
```

### Notes

By default Helm doesn't save generated K8s yaml files to disk. If you want to see what helm is generating you can use the following command to output the K8s yamls:

```
helm template {helm-app-name}
```

```
helm create my-super-app
```

```
helm template my-super-app
```

## 11.16 Installing A Custom Chart

- Custom charts are created in the current directory where Helm can find and install them just like any other chart

```
helm create my-super-app
helm install msapp my-super-app
```

- The 'helm create' command creates a chart that is complete and usable as-is. It is meant to be customized but can be installed and used on Kubernetes without any customization (for example for testing purposes).

## 11.17 Packaging Custom Charts

- Custom charts can also be packaged to an archive and installed from there:

```
helm package my-super-app # creates my-super-app-0.1.0.tgz
helm install msa my-super-app-0.1.0.tgz
```

- Helm packages can be signed using this form of the command:

```
helm package {chart-path} --key {key} --keyring {keyring-filespec}
```

- More information about the package command can be found here:

[https://helm.sh/docs/helm/helm\\_package/](https://helm.sh/docs/helm/helm_package/)

## Notes

### 11.18 Summary

In this chapter we covered:

- What is Helm?
- Installing Helm
- Helm Features
- Helm Terminology
- Searching for Charts
- Adding Repositories
- Installing Charts
- Upgrading and Rolling back Releases
- Creating Custom Charts
- Helm Templates
- Packaging and Installing Custom Charts



## Chapter 12 - Logging, Monitoring, and Troubleshooting

---

### *Objectives*

Key objectives of this chapter

- Logging in Kubernetes
- Monitoring Kubernetes
- Debugging Kubernetes
- Upgrading Kubernetes

### **12.1 Differences Between Logging and Monitoring**

- Logging records events (e.g., a Pod being created, and monitoring records statistics (e.g., the latency of a particular request, the CPU used by a process, or the number of requests to a particular endpoint).
- Logged records, by their nature, are discrete, whereas monitoring data is a sampling of some continuous value.
- Logging systems are generally used to search for relevant information. (e.g. pod failures) For this reason, log storage systems are oriented around storing and querying vast quantities of data
- Monitoring systems are generally geared around visualization. (e.g. CPU utilization over a period of time) Thus, they are stored in systems that can efficiently store time-series data.
- Monitoring data can give you a good sense of the overall health of your cluster and can help you identify anomalous events that may be occurring.
- Logging, on the other hand, is critical for diving in and understanding what is happening, possibly across many machines, to cause such anomalous behavior.

### **12.2 Logging in Kubernetes**

- Logs are useful for debugging problems and monitoring cluster activity.
- The easiest and most embraced logging method for containerized applications is to write to the standard output and standard error streams.

- However, the native functionality provided by a container engine or runtime is usually not enough for a complete logging solution.
- For example, if a container crashes, a pod is evicted, or a node dies, you'll usually still want to access your application's logs.
- Logs should have separate storage and lifecycle independent of nodes, pods, or containers.
- This concept is called cluster-level-logging.
- Cluster-level logging requires a separate backend to store, analyze, and query logs.
- Kubernetes provides no native storage solution for log data, but you can integrate many existing logging solutions into your Kubernetes cluster.

## 12.3 Basic Logging

- To read messages logged by containerized apps, use the following command:

```
kubectl logs <pod_name>
```

- To view logs of a crashed container instance, you can use the following command:

```
kubectl logs <pod_name> -previous
```

- This command fetches log entries from the container log file.
- If the container is killed and then restarted by Kubernetes, you can still access logs from the previous container.
- if the pod is evicted from the node, log files are lost.

## 12.4 Logging Agents

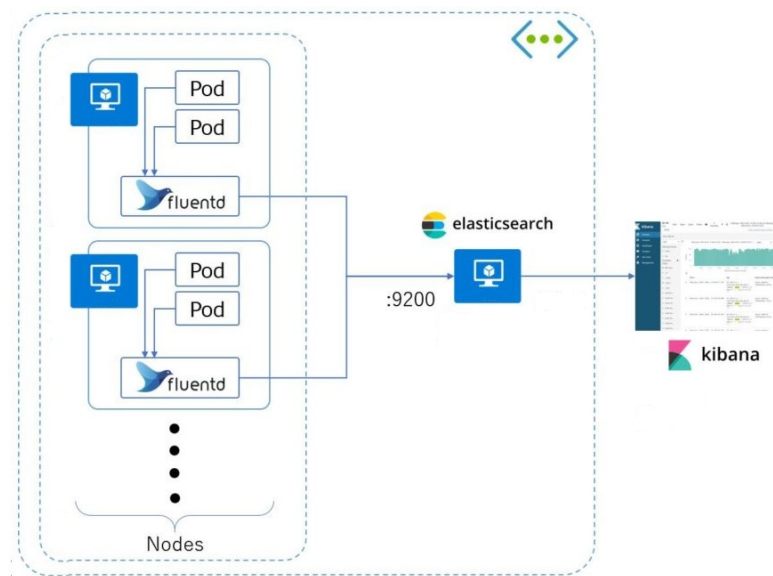
- Logging agents expose K8S logs and push them to a configured location.
- Typically, these agents are containers that have access to a directory with logs from all applications on the node.
- This solution ensures the logs are still available even if a pod or node is no longer available.



- Every pod in a K8S cluster has its standard output and standard error captured and stored in the `/var/log/containers/` node directory.
- A logging agent is a DaemonSet type of pod that exists on each cluster node
- A logging agent captures the logs provided in each node's `/var/log/containers/` directory and processes them somehow.

## 12.5 Fluentd and Elastic Stack

- Fluentd is a logging agent that unifies the collection and consumption of data in a pluggable manner.
- It is deployed to each node in the Kubernetes cluster, which will collect each container's log files running on that node.

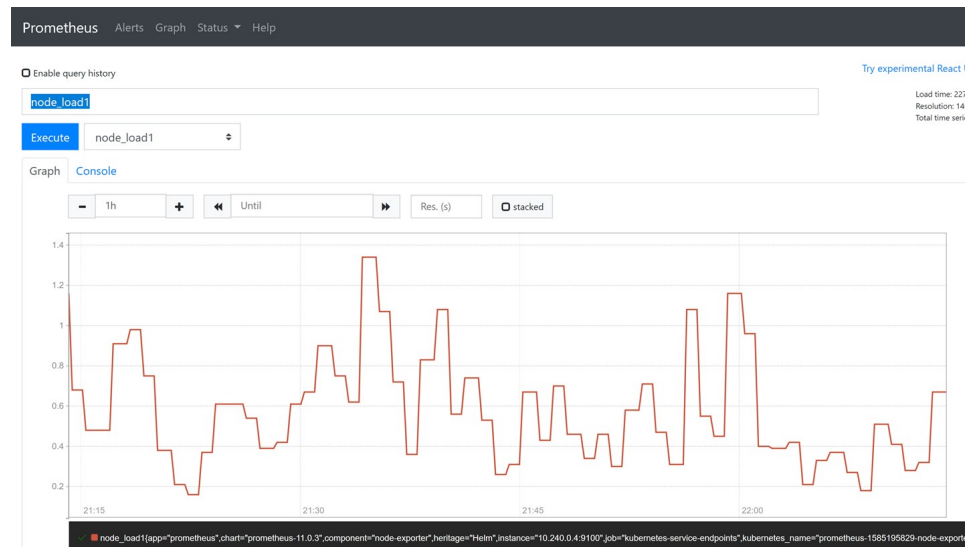


source: [https://miro.medium.com/max/1794/1\\*eNDJDq\\_eWief-XfK41lMRg.png](https://miro.medium.com/max/1794/1*eNDJDq_eWief-XfK41lMRg.png)

- K8S acts as a log producer, Fluentd acts as a log collector, and Elastic Stack acts as a log consumer.
- To allow these components to work together, a fluentd configuration file maps how the input data will be found and processed and where it will be saved.

## 12.6 Monitoring with Prometheus

- Prometheus is an open-source monitoring and alerting toolkit made for monitoring applications in clusters.
- Kubernetes features built-in support for Prometheus metrics and labels as well
- Prometheus runs as a pod in your Kubernetes cluster.
- Deploying Prometheus requires two Kubernetes objects:
  - ◇ a Deployment for the Prometheus pod itself
  - ◇ a ConfigMap that tells Prometheus how to connect to your cluster. This manifest file contains descriptions of both objects.



## 12.7 Kubernetes and Prometheus - Metrics

- You can monitor various metrics, such as:
  - ◇ container\_memory\_usage\_bytes
  - ◇ api\_server\_request\_count
  - ◇ ...
- For example:

```
container_memory_usage_bytes{image="CONTAINER:VERSION"}
```

- You can also use regular expressions when querying metrics:
- For example:

```
container_memory_usage_bytes{image=~"CONTAINER:.*"}  
  ◇ =~ signifies it's a regular expression
```

## 12.8 Alerting

- Alerts can be added after monitoring is set up.
- Example of alerts includes:
  - ◇ **Whitebox:** consumption of CPU, memory, and storage
  - ◇ **Blackbox:** the latency of a request to the API server or the number of 403 (Unauthorized) responses that your API server is returning.
- Whitebox alerting offers a critical heads-up before significant problems occur.
- Blackbox alerting, on the other hand, gives you high-quality alerts caused by real, user-facing problems.

## 12.9 Debugging Pods

- The most basic way to check a pod status is to use the following command:

```
kubectl get pods
```

- You can retrieve a lot of information about each of the pods using:

```
kubectl describe pod <pod_name>
```

- The command shows configuration information about the container(s) and Pod (labels, resource requirements, etc.), as well as status information about the container(s) and Pod (state, readiness, restart count, events, etc.).
- The container state is one of Waiting, Running, or Terminated.
- Depending on the state, additional information will be provided

## 12.10 Debugging Pods (Contd.)

- If the pod state is running, the system tells you when the container started.
- Ready tells you whether the container passed its last readiness probe.
- Restart Count tells you how many times the container has been restarted
- Lastly, the command shows a log of recent events related to your Pod.
- To get more event details, use the following command:

```
kubectl get events
```

- Alternate to kubectl describe pod command is to use the following command:

```
kubectl get pod <pod_name> -o yaml
```

## 12.11 Debugging Nodes

- The basic way to find nodes status is to use the following command:

```
kubectl get nodes
```

- To get more details, you can use the following command:

```
kubectl describe node <node_name>
```

- Alternatively, to view the details in yaml format, use the following command:

```
kubectl get node <node_name> -o yaml
```

## 12.12 Debugging Replication Controllers and Services

- You can also use the following command to inspect events related to the replication controller

```
kubectl describe rc ${CONTROLLER_NAME}
```

- To troubleshoot services, ensure the service endpoints are created:

```
kubectl get endpoints <service_name>
```

## 12.13 Upgrading Kubernetes

- The upgrade workflow at a high-level is the following:

- ◇ Upgrade the primary control plane node.
- ◇ Upgrade worker nodes.
- Kubernetes Control Plane is composed of Kubernetes Master and kubelet processes.
- The Control Plane maintains a record of all of the Kubernetes Objects in the system and runs continuous control loops to manage those objects' state
- Make sure to back up any important components, such as app-level state stored in a database. Even though the upgrade process does not touch your workloads, only components internal to Kubernetes, but backups are always a best practice
- All containers are restarted after the upgrade, because the container spec hash value is changed.
- You only can upgrade from one MINOR version to the next MINOR version, or between PATCH versions of the same MINOR.
  - ◇ You cannot skip MINOR versions when you upgrade. For example, you can upgrade from 1.y to 1.y+1, but not from 1.y to 1.y+2.

## 12.14 Upgrade Process

- Determine which version to upgrade to
- Upgrade control plane node
  - ◇ Upgrade kubeadm
  - ◇ Get upgrade plan
  - ◇ Apply the upgrade
  - ◇ Upgrade kubelet and kubectl
  - ◇ Restart kubelet
- Upgrade worker nodes
  - ◇ Upgrade kubeadm
  - ◇ Drain the node/prepare the node for maintenance

- ◇ Upgrade kubelet configuration
- ◇ Upgrade kubelet and kubect
- ◇ Restart kubelet
- ◇ Uncordon the node/bring the node online

## 12.15 Determine Which Version to Upgrade To

- Ubuntu, Debian

```
apt update
apt-cache policy kubeadm
```

- CentOS, RHEL

```
yum list --showduplicates kubeadm --
disableexcludes=kubernetes
```

## 12.16 Upgrade kubeadm

- kubeadm should be upgraded on the control plane and all worker nodes
- Upgrade it on control plane node first followed by the worker nodes
- Ubuntu, Debian

```
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=<version>
&& \
apt-mark hold kubeadm
```

- CentOS, RHEL

```
yum install -y kubeadm-<version> --
disableexcludes=kubernetes
```

## 12.17 Upgrade Control Plane Node

- Get the upgrade plan

```
sudo kubeadm upgrade plan
```

- It will display output like this: (Your versions may vary)

COMPONENT	CURRENT	AVAILABLE
-----------	---------	-----------

**Kubelet**      1 x v1.14.2      v1.15.0

- Apply the upgrade

```
sudo kubeadm upgrade apply <version>
```

## 12.18 Upgrade kubelet and kubectl

- Ubuntu, Debian

```
apt-mark unhold kubelet kubectl && \
```

```
apt-get update && apt-get install -y kubelet=<version>  
kubectl=<version> && \
```

```
apt-mark hold kubelet kubectl
```

- CentOS, RHEL

```
yum install -y kubelet-<version> kubectl-<version> --  
disableexcludes=kubernetes
```

- Restart the kubelet

```
sudo systemctl restart kubelet
```

## 12.19 Upgrade Worker Nodes

- The upgrade procedure on worker nodes should be executed one node at a time or few nodes at a time, without compromising the minimum required capacity for running your workloads
- Drain the node i.e. prepare the node for maintenance by marking it unschedulable and evicting the workload

```
kubectl drain $NODE --ignore-daemonsets
```

- Upgrade kubelet configuration

```
sudo kubeadm upgrade node
```

- Upgrade kubelet and kubectl on worker nodes (already covered in one of the previous slides)
- Restart the kubelet (already covered in one of the previous slides)

- Uncordon or bring the node online by marking it schedulable

```
kubect1 uncordon $NODE
```

## 12.20 Recovering From a Failure State

- If kubeadm upgrade fails and does not roll back, for example because of an unexpected shutdown during execution, you can run the following command again:

```
kubeadm upgrade
```

- The command is idempotent and eventually makes sure that the actual state is the desired state you declare.
- To recover from a bad state, you can also run the following command:

```
kubeadm upgrade --force
```

- The command recovers a node without changing the version that your cluster is running

## 12.21 Summary

- Logging and monitoring are critical components of understanding how your cluster and your applications are performing
- Centralized logging and monitoring are not part of Kubernetes. You can use various logging and monitoring solutions, such as ELK (Elasticsearch, Logstash, Kibana), EFK (Elasticsearch, Fluentd, Kibana), and Splunk
- Kubernetes can be upgraded to a newer version



## Chapter 13 - Continuous Integration Fundamentals

---

### *Objectives*

Key objectives of this chapter

- Jenkins
- Installing
- Jobs
- SCM Integration
- Build Steps
- Pipelines
- Plugins

### **13.1 Jenkins Continuous Integration**

- Originally developed at Sun by Kohsuke Kawaguchi?
  - ◇ Originally “Hudson” on java.net circa 2005
  - ◇ Jenkins forked in November 2010
  - ◇ Hudson is still live, part of Eclipse Foundation
  - ◇ But Jenkins seems to be far more active

### **13.2 Jenkins Features**

- Executes jobs based on a number of triggers
  - ◇ Change in a version control system
  - ◇ Time
  - ◇ Manual Trigger
- A Job consists of some instructions
  - ◇ Run a script
  - ◇ Execute a Maven project or Ant File

- ◇ Run an operating system command
- User Interface can gather reports
  - ◇ Each job has a dashboard showing recent executions

## 13.3 Running Jenkins

- You can run Jenkins Standalone or inside a web container
- You can set up distributed instances that cooperate on building software
- Can set up jobs in place of what might have been script commands.

## 13.4 Downloading and Installing Jenkins


- Download Jenkins from the Jenkins website (<http://jenkins-ci.org>)
  - Jenkins is a dynamic project, and new releases come out at a regular rate.
- Jenkins distribution is bundled in Java web application (a WAR file).
- Windows users, there is a graphical Windows installation MSI package for Jenkins.



# Jenkins

An extendable open source continuous integration server

[BLOG](#) [CONNECT](#) [BUG TRACKER](#) [WIKI](#) [CI](#) [TUTORIALS](#) [ARCHIVES](#) [DONATION](#) [ABOUT](#)



### Meet Jenkins

Find out what Jenkins is and get started.

### Download Jenkins

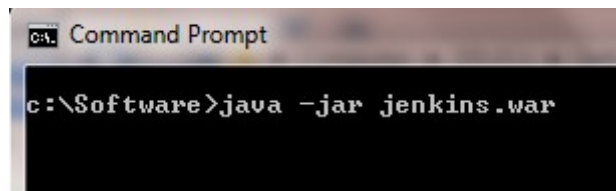
Release Long-Term Support Release

**Java Web Archive (.war)**  
**Latest and greatest (1.539)**  
[changelog](#) | [past releases](#) | [RC](#)  
[upgrading from Hudson?](#)

Native packages

## 13.5 Running Jenkins as a Stand-Alone Application

- Jenkins comes bundled as a WAR file that you can run directly using an embedded servlet container.
- Jenkins uses the lightweight Servlet engine to allow you to run the server out of the box.
- Flexible to install plug-ins and upgrades on the fly.
- To run Jenkins using the embedded Servlet container, just go to the command line and type the following:



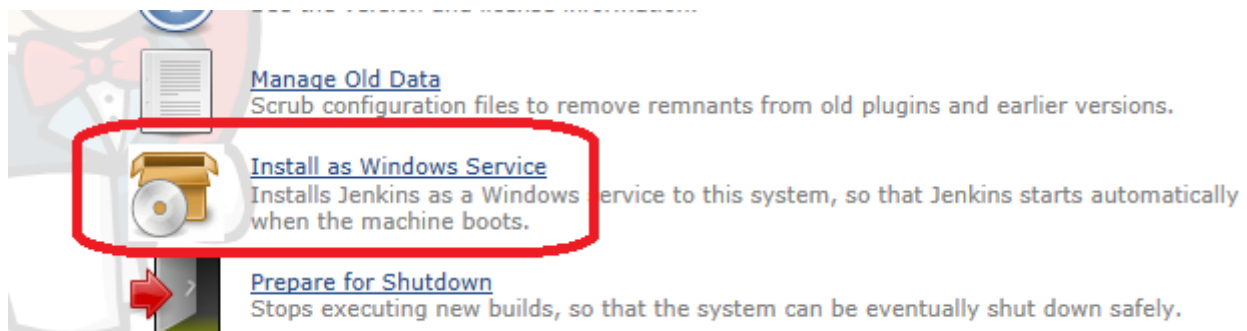
- The Jenkins web application will now be available on port 8080.
- Jenkins can be accessed directly using the server URL (<http://localhost:8080>).
- To stop Jenkins, just press Ctrl-C.
- Useful Options:
  - `--httpPort`
    - By default, Jenkins will run on the 8080 port.
    - Jenkins can be started on a different port using the `--httpPort` option:
      - `java -jar jenkins.war --httpPort=8081`
  - `--logfile`
    - By default, Jenkins writes its log file into the current directory.
    - Option to redirect your messages to another file:
      - `java -jar jenkins.war --logfile=C:/Software/log/jenkins.log`
- These options can also be set at `JENKINS_HOME/jenkins.xml` config file.

## 13.6 Running Jenkins on an Application Server

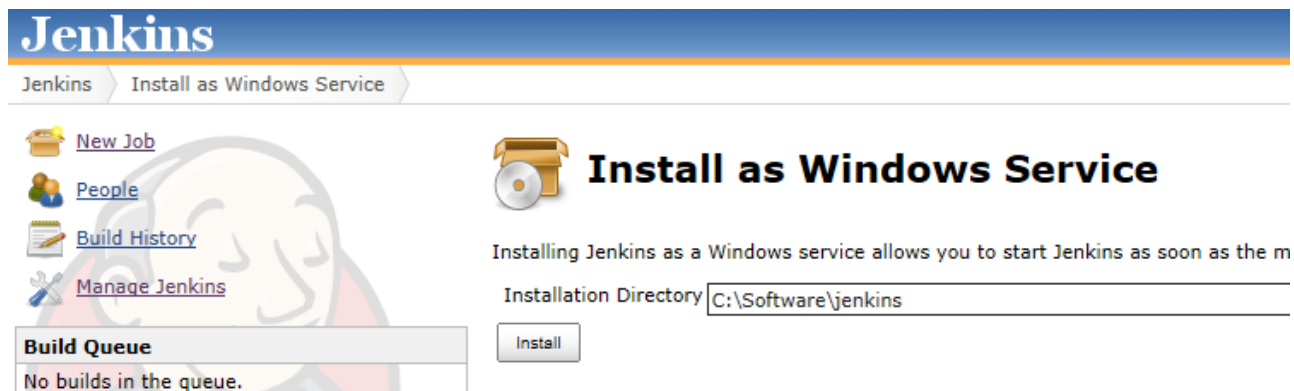
- Jenkins distribution WAR file can be easily deployed to standard Java application server such as Apache Tomcat, Jetty, or GlassFish.
- Jenkins will be executed in its own web application context (typically "jenkins").
  - URL: <http://localhost:8080/jenkins>.

## 13.7 Installing Jenkins as a Windows Service

- In production, installation of Jenkins on a Windows box is essential to have it running as a Windows service.
  - Jenkins will automatically start whenever the server reboots
  - Can be managed using the standard Windows administration tools.
- Jenkins using the windows installer automatically runs Jenkins as a windows service.
  - No need to do anything.
- First, Start the Jenkins server on your machine:
  - `java -jar jenkins.war`
- Connect to Jenkins by going to the following URL `http://<hostname>:8080`
- Look for "Install as Windows Service" link in the "Manage Jenkins" page.



- Clicking this link shows you the installation screen:



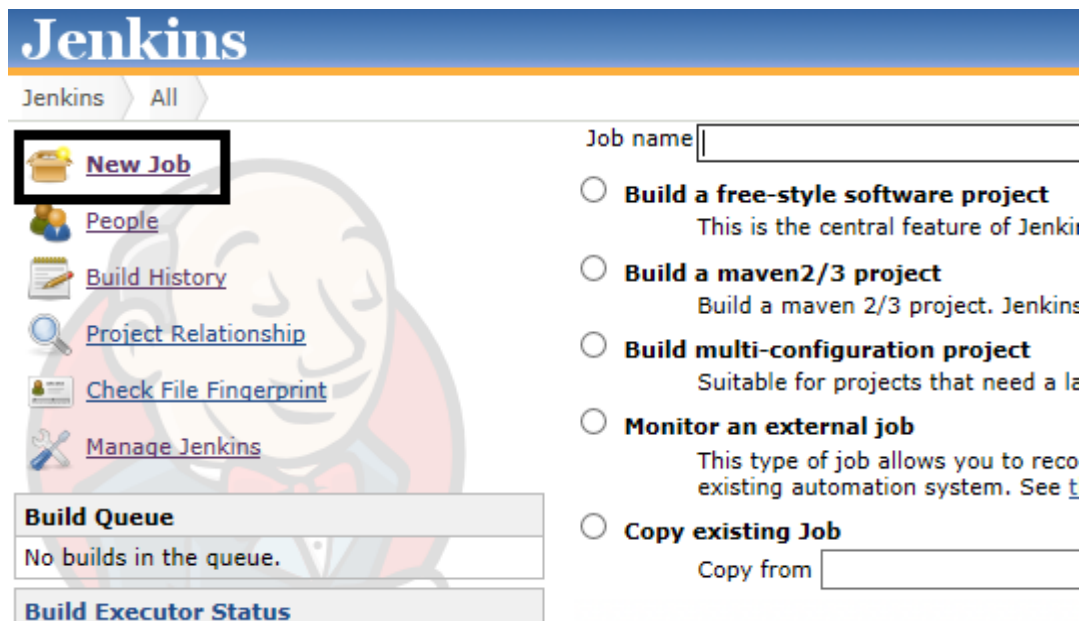
- Choose a directory where Jenkins will be installed, JENKINS\_HOME and used to store data files and programs
- Upon successful completion of the installation, you should see a page asking you to restart Jenkins.
- At this point, you can use the service manager to confirm that Jenkins is running as a service.

Name	Description	Status	Startup Type	Log On As
Indexing Service	Indexes co...		Manual	Local System
IPSEC Services	Manages I...	Started	Automatic	Local System
Java Quick Starter	Prefetches...	Started	Automatic	Local System
jenkins	This servic...	Started	Automatic	Local System
Real-time Disk Monitoring	Rebates...	Started	Automatic	Local System

## 13.8 Different types of Jenkins job

- Jenkins supports several different types of build jobs
  - Freestyle software project:
    - Freestyle projects are general purpose and allow to configure any sort of build job.
    - Highly flexible and very configurable.
  - Maven project:
    - Jenkins understands Maven pom files and project structures.

- Reduce the work needed to do to set up the project.
- Monitor an external job:
  - Monitoring the non-interactive execution of processes, such as cron jobs.
- Multi-configuration job:
  - Run same build job in many different configurations.
  - Powerful feature, useful for testing an application in many different environments.



## 13.9 Configuring Source Code Management(SCM)

- Monitors version control system, and checks out the latest changes as they occur.
- Compiles and tests the most recent version of the code.
- Simply check out and build the latest version of the source code on a regular basis.
- SCM configuration options in Jenkins are identical across all sorts of build jobs.

- Jenkins supports CVS and Subversion out of the box, with built-in support for Git
- Integrates with a large number of other version control systems via plugins.

### **13.10 Working with Subversion**

- Simply provide the corresponding Subversion URL
  - Supported protocols HTTP, svn, or file.
- Jenkins will check that the URL is valid as soon as you enter it.
- If authentication needed, Jenkins will prompt you for the corresponding credentials automatically, and store them for any other build jobs that access this repository.
- Fine-tune Jenkins to obtain the latest source code from your Subversion repository by selecting an appropriate value in the Check-out Strategy drop-down list.
- Choose check-out Strategy as “Use ‘svn update’ as much as possible, with ‘svn revert’ before update”
  - No local files are modified, though it will not remove any new files that have been created during the build process.
  - You might want other options, depending on the load on your svn server.

## 13.11 Working with Subversion (cont'd)

### Source Code Management

---

☐ CVS

☐ None

☒ Subversion

Modules

Repository URL



Repository URL is required.

Local module directory (optional)

Check-out Strategy

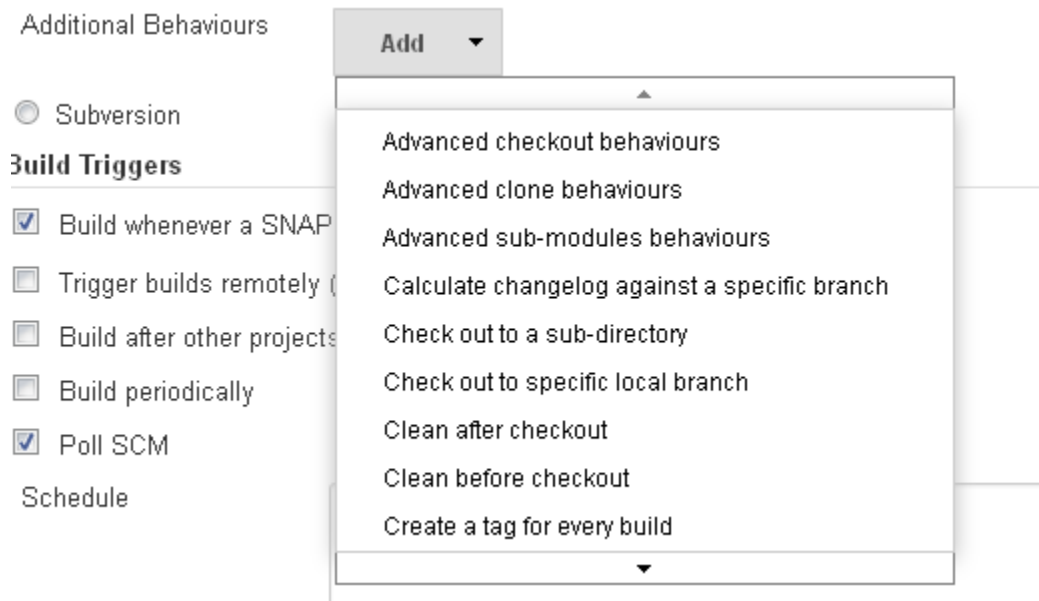
Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository browser

## 13.12 Working with Git

- Oddly enough, Git support isn't enabled by default
- Enable the 'Git Plugin' through the plugin management screen
- Any type of remote repository can be used:
  - ◇ Could be https, ssh, or local
- Jenkins will check that the URL is valid as soon as you enter it.
- You can store ssh credentials or HTTP credentials
- There are a wide variety of other "Additional Checkout Behaviors" that are possible






### 13.13 Build Triggers

- In a Freestyle build, there are three basic ways a build job can be triggered
  - Start a build job once another build job has completed
  - Kick-off builds at periodical intervals
  - Poll the SCM for changes

**Build Triggers**

☒ Build after other projects are built  
Project names

 **No project specified**  
Multiple projects can be specified like 'abc, def'

☒ Build periodically  
Schedule

☒ Poll SCM  
Schedule

Ignore post-commit hooks ☐

### 13.14 Schedule Build Jobs

- Build job at regular intervals.
- For all scheduling tasks, Jenkins uses a cron-style syntax, consisting of five fields separated by white space in the following format:
  - ◇ MINUTE : Minutes within the hour (0–59)
  - ◇ HOUR : The hour of the day (0–23)
  - ◇ DOM : The day of the month (1–31)
  - ◇ MONTH : The month (1–12)
  - ◇ DOW : The day of the week (0–7) where 0 and 7 are Sunday.
- There are also a few short-cuts:
  - ◇ “\*” represents all possible values for a field. For example, “\* \* \* \* \*” means “once a minute.”
  - ◇ You can define ranges using the “M–N” notation. For example “1-5” in the DOW field would mean “Monday to Friday.”
  - ◇ You can use the slash notation to defined skips through a range. For

example, “\*/5” in the MINUTE field would mean “every five minutes.”

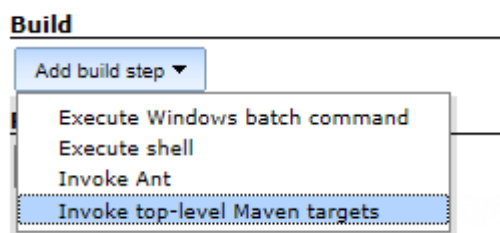
- ◇ A comma-separated list indicates a list of valid values. For example, “15,45” in the MINUTE field would mean “at 15 and 45 minutes past every hour.”

### 13.15 Polling the SCM

- Poll SVN server at regular intervals if any changes have been committed.
- Jenkins kicks off a build, source code in the project.
- Polling frequently to ensure that a build kicks off rapidly after changes have been committed.
- The more frequent the polling is, the faster the build jobs will start, and the more accurate.
- In Jenkins, SCM polling is easy to configure and uses the same cron syntax we discussed previously.

### 13.16 Maven Build Steps

- Jenkins has excellent Maven support, and Maven build steps are easy to configure and very flexible.
- Select “Invoke top-level Maven targets” from the build step lists.



- Select a version of Maven to run (if you have multiple versions installed)
- Enter the Maven goals you want to run. Jenkins freestyle build jobs work fine with both Maven 2 and Maven 3.
- The optional POM field lets you override the default location of the Maven pom.xml file.

### **Build**

Invoke top-level Maven targets

Maven Version

Goals

## **13.17 Configuring Jenkins to Access Kubernetes**

- In Jenkins, create an **Environment** variable:

**Name:** KUBECONFIG

**Value:** \$HOME/.kube/config

## **13.18 Jenkins Pipeline**

- Same Pipeline code

```
node {
  stage ('Checkout') {
    git url: '/var/lib/jenkins/repos/hello-node'
  }
  stage ('Build Docker Image') {
    sh 'docker build -t node-app:v1.0 .'
  }
  stage ('Delete Deployment') {
    sh 'kubectl delete deployment myproject --ignore-not-found=true'
  }
  stage ('Create Deployment myproject') {
    sh 'kubectl create deployment myproject --image="node-app:v1.0"'
  }
  stage ('Delete Service') {
    sh 'kubectl delete service myproject --ignore-not-found=true'
  }
  stage ('Create Service') {
    sh 'kubectl expose deployment myproject --type=NodePort --port=9090'
  }
}
```

## 13.19 Jenkins Pipeline Output

### Stage View

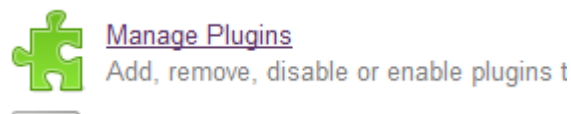


## 13.20 Installing Jenkins Plugins

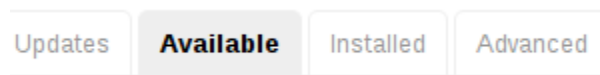
- For example, installing the Jenkins Emma plugin for code coverage.



- Click **Manage Jenkins**
- Then on click the **Manage Plugins** link:



- Select the Available tab:



- For quick access, enter plugin name on the filter input box, located at the

top-right corner.

- Select the Restart Jenkins when the installation is complete and no jobs are running.
- You have done the installation
- Note that quite often, the "Install without restart" option works too

## **13.21 Summary**

In this chapter we covered:

- Jenkins
- Installing
- Jobs
- SCM Integration
- Build Steps
- Pipelines
- Plugins